FRET 41
FRONTIERS IN ELECTRONIC TESTING

Michael Nicolaidis
Editor

Soft Errors in Modern Electronic Systems

Springer

VISIT…

Frontiers in Electronic Testing

FRONTIERS IN ELECTRONIC TESTING

Consulting Editor: **Vishwani D. Agrawal**

Volume 41

For further volumes
http://www.springer.com/series/5994

Michael Nicolaidis
Editor

Soft Errors in Modern Electronic Systems

Springer

Editor
Dr. Michael Nicolaidis
TIMA Laboratory
Grenoble INP, CNRS, UJF
av. Felix Viallet 46
38031 Grenoble CX
France
michael.nicolaidis@imag.fr

ISSN 0929-1296
ISBN 978-1-4419-6992-7 e-ISBN 978-1-4419-6993-4
DOI 10.1007/978-1-4419-6993-4
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2010933852

© Springer Science+Business Media, LLC 2011
All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.
The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

The ideas of reliability, or should I say unreliability, in computing began with von Neumann's 1963 paper [1]. In the intervening years, we flip-flopped between thoughts such as "semiconductors are inherently reliable" and "increasing complexity can lead to error buildup".

Change over to digital technology was a welcome relief from a variety of electrical noises generated at home. While we continue to fictionalize the arrival of extraterrestrial beings, we did not suspected that they would arrive early to affect our electronic systems. Let me quote from a recent paper, "From the beginning of recorded history, man has believed in the influence of heavenly bodies on the life on the Earth. Machines, electronics included, are considered scientific objects whose fate is controlled by man. So, in spite of the knowledge of the exact date and time of its manufacture, we do not draft a horoscope for a machine. Lately, however, we have started noticing certain behaviors in the state of the art electronic circuits whose causes are traced to be external and to the celestial bodies outside our Earth [2]".

May and Woods of Intel Corporation reported on alpha particle induced *soft errors* in the 2107-series 16-KB DRAMs. They showed that the upsets were observed at sea level in dynamic RAMs and CCDs. They determined that these errors were caused by α particles emitted in the radioactive decay of uranium and thorium present just in few parts per million levels in package materials. Their paper represents the first public account of radiation-induced upsets in electronic devices at the sea level and those errors were referred to as "soft errors" [3].

It has been recognized since 1940s that an *electromagnetic pulse* (EMP) can cause temporal malfunction or even permanent damage in electronic circuits. The term EMP refers to high energy electromagnetic radiation typically generated by lightning or through interaction of charged particles in the upper atmosphere with γ rays or X rays. Carl E. Baum, perhaps the most significant contributor to the EMP research, traces the history of the EMP phenomenon and reviews a large amount of published work in his 188-reference survey article [4]. Besides providing techniques of radiation hardening, shielding and fault-tolerance, significant amount of experimental work has been done on developing EMP simulator hardware. I particularly mention this because I believe that collaboration between soft error and EMP research communities is possible and will be beneficial.

The publication of this book is the latest event in the history I have cited above. Its contributing editor, Michael Nicolaidis, is a leading authority on soft errors. He is an original contributor to research and development in the field. Apart from publishing his research in a large number of papers and patents he cofounded iROC Technologies. His company provides complete soft-error analysis and design services for electronic systems.

Nicolaidis has gathered an outstanding team of authors for the ten chapters of this book that cover the breadth and depth. This is the first book to include almost all aspects of soft errors. It comprehensively includes historical views, future trends, the physics of SEU mechanisms, industrial standards and practices of modeling, error mitigation methods, and results of academic and industry research. There is really no other published book that has such a complete coverage of soft errors.

This book fills a void that has existed in the technical literature. In the words of my recently graduated student, Fan Wang, "During the time I was a graduate student I suffered a lot trying to understand different topics related to soft errors. I have read over two hundred papers on this topic. Soft error is mentioned in most books on VLSI reliability, silicon technology, or VLSI defects and testing, however, there is no book specifically on soft errors. Surprisingly, the reported measurements and estimated results in the scattered literature vary a lot sometimes even seem to contradict each other. I believe this book will be very useful for academic research and serve as an industry guide".

The book provides some interesting reading. The early history of soft errors is like detective stories. Chapter 1 documents the case of soft errors in the Intel 2107-series 16-kb DRAMs. Culprits are found to be alpha particles emitted through the radioactive decay of uranium and thorium impurities in the packaging material. The 1999 case of soft errors in Sun's Enterprise server results in design reforms leading to the applications of coding theory and inventions of new design techniques.

A serious reader must go through Chap. 2 to learn the terms and definitions and Chap. 3 that provides the relevant standards. Chapters 4 and 5 discuss methodologies for modeling and simulation at gate and system levels, respectively. Hardware fault injection techniques are given in Chap. 6, with accelerated testing discussed in Chap. 7. Chapters 8 and 9 deal with soft-error mitigation techniques at hardware and software levels, respectively. Chapter 10 gives techniques for evaluating the soft-error tolerance of systems.

Let us learn to deal with soft errors before they hurt us.

Vishwani D. Agrawal

References

1. J. von Neumann, "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components (1959)", in A. H. Taub, editor, *John von Neumann: Collected Works, Volume V: Design of Computers, Theory of Automata and Numerical Analysis*, Oxford University Press, 1963, pp. 329–378.

2. F. Wang and V. D. Agrawal, “Single Event Upset: An Embedded Tutorial”, in *Proc. 21st International Conf. VLSI Design*, 2008, pp. 429–434.
3. T. C. May and M. H. Woods, “A New Physical Mechanism for Soft Errors in Dynamic Memories”, in *Proc. 16th Annual Reliability Physics Symposium*, 1978, pp. 33–40.
4. C. E. Baum, “From the Electromagnetic Pulse to High-Power Electromagnetics”, *Proceedings of the IEEE*, vol. 80, no. 6, 1992, pp. 789–817.

Preface

In the early computer era, unreliable components made fault-tolerant computer design mandatory. Dramatic reliability gains in the VLSI era restricted the use of fault-tolerant design in critical applications and hostile environments. However, as we are approaching the ultimate limits of silicon-based CMOS technologies, these trends have been reversed. Drastic device shrinking, very low operating voltages, increasing complexities, and high speeds made circuits increasingly sensitive to various kinds of failures. Due to these trends, soft errors, considered in the past as a concern for space applications, became during the past few years a major source of system failures of electronic products even at ground level. Consequently, soft-error mitigation is becoming mandatory for an increasing number of application domains, including networking, servers, avionics, medical, and automotive electronics. To tackle this problem, chip and system designers may benefit from several decades of soft error related R&D from the military and space. However, as ground-level applications concern high-volume production and impose stringent cost and power dissipation constraints, process-based and massive-redundancy-based approaches used in military and space applications are not suitable in these markets.

Significant efforts have therefore been made during the recent years in order to benefit from the fundamental knowledge and engineering solutions developed in the past and at the same time develop new solutions and tools for supporting the constraints of ground-level applications. After design for test (DFT), design for manufacturability (DFM), and design for yield (DFY), the design for reliability (DFR) paradigm is gaining importance starting with design for soft error mitigation. Dealing with soft errors is a complex task that may involve high area and power penalties, as copying with failures occurring randomly during system operation may require significant amounts of redundancy. As a consequence, a compendium of approaches is needed for achieving product reliability requirements at low area and power penalties. Such approaches include:

- Test standards for characterizing the soft-error rate (SER) of the final product and of circuit prototypes in the terrestrial environment. Such standards are mandatory for guarantying the accuracy of test results and for having a common

reference when comparing the SER, measured in terms of Failure in Time (FIT), of devices provided by different suppliers.

- SER-accelerated testing platforms, approaches, and algorithms for different devices, including SRAMs, DRAMs, TCAMs, FPGAs, processors, SoCs, and complete systems.
- SER-accelerated testing platforms, approaches, and algorithms for cell libraries.
- Software/hardware methodologies and tools for evaluating SER during the design phase. Such tools become increasingly important for two major reasons. Characterizing SER during the design phase is the only way for avoiding bad surprises when the circuit prototype or the final product is tested, which could lead to extra design and fabrication cycles and loss of market opportunities. Interactive SER estimation during the design cycle is the only way for making the necessary tradeoffs, determining the circuit critical parts and selecting the most efficient mitigation approaches for meeting a reliability target at minimal cost in terms of power, speed, and area. Software and hardware tools at various levels are required such as:
  - TCAD tools for characterizing the transient current pulses produced by alpha particles and secondary particles.
  - Cell FIT estimation tools to guide the designers of memory cells and cell libraries for meeting their SER budget at minimal cost, and for providing the cell FIT to the higher level SER estimation tools.
  - Spice-level FIT estimation, usually for evaluating the impact of transient pulses in sequential cells and in combinational logic.
  - Gate-level FIT estimation tools for characterizing IP blocks: based on exact, statistical or probabilistic approaches; considering the logic function only (for logic derating computation) or both the logic function and the SDF files (for combined logic and time derating computation).
  - RTL FIT estimation.
  - SoC FIT estimation, for taking into account the functional derating at the SoC level.
  - Fault injection in hardware platforms for accelerating the FIT estimation task at IP level and SoC level.
- Soft-error mitigation approaches at hardware level including: error detecting and correcting codes, hardened cells, self-checking circuits, double sampling approaches, instruction-level retry.
- Soft-error mitigation approaches at software level, operating system level, as well as check-pointing and rollback recovery.

Grenoble, France — Michael Nicolaidis
February 2010

Purpose

The purpose of this book is to provide a comprehensive description of the highly complex chain of physical processes that lead to the occurrence of soft errors. Mastering soft errors and the related chain of physical processes requires mastering numerous technological domains, including: nuclear reactions of cosmic rays with the atmosphere (neutron and proton generation at ground level); nuclear reactions of atmospheric neutrons and protons with die atoms (secondary particles generation); coulomb interaction (ionization); device physics (charge collection); electrical simulation (SEU creation, SET propagation); event-driven simulation (for combined logic and time derating estimation); logic domain simulation (for logic derating estimation); RTL simulation; and hardware emulation. Most of these domains are extremely challenging and may lead to unacceptable simulation time for achieving good accuracy. Past and recent developments in these domains are reported in the book.

The book is also aimed at providing a comprehensive description of various hardware and software techniques enabling soft-error mitigation at moderate cost. This domain is also challenging, since coping with failures occurring randomly during system operation is not trivial and usually requires significant amounts of redundancy. Recent developments dealing with these issues are also reported.

Finally, as other reliability threats, including variability, EMI and accelerating aging are gaining significance, solutions that could be used to simultaneously address all of them are also discussed.

To reach its goals, the book is organized in ten chapters following a coherent sequence, starting with Chap. 1:

- *Soft Errors, from Space to Ground: Historical Overview, Empirical Evidence and Future Trends*

and finishing with Chap. 10:

- *Specification and Verification of Soft Error Performance in Reliable Electronic Systems, dealing with Soft-Errors in Complex Industrial Designs*

through eight chapters dealing with:

- *Single Event Effects: Mechanisms and Classification*
- *JEDEC Standards on Measurement and Reporting of Alpha Particles and Terrestrial Cosmic Ray-Induced Soft Errors*
- *Gate Level Modeling and Simulation*
- *Circuit and System Level Single Event Effects Modeling and Simulation*
- *Hardware Fault Injection*
- *Integrated circuit qualification for Space and Ground-level Applications: Accelerated tests and Error-Rate Predictions*
- *Circuit-level Soft-Error Mitigation*
- *Software-Level Soft-Error Mitigation Techniques*

The aim of this volume is to be a reference textbook describing: all the basic knowledge on soft errors for senior undergraduate students, graduate students in MSc or PhD tracks and teachers; the state-of-the-art developments and open issues in the field for researchers and professors conducting research in this domain; and a comprehensive presentation of soft errors-related issues and challenges that may face circuit and system designers and managers, together with the most efficient solutions, methodologies, and tools that they can use to deal with.

Grenoble, France Michael Nicolaidis
March 2010

Acknowledgments

The authors of the chapters have devoted a significant amount of efforts and passion in describing complex technical problems in a clear, understandable, comprehensive, and still concise manner, without sacrificing accuracy. Tino, Remi, Charles, Nadine, Lorena, Dan, Luis, Celia, Mario, Marta, Raoul, Gilles, Paul, Maurizio, Matteo, Massimo, Allan, Adrian, Ana, Shi-Jie, David, Ron, Dean and Ian, I would like to warmly thank all of you for doing your best in providing high quality chapters that enhance the overall quality of the book.

I would like to thank Prof. Vishwani D. Agrawal for making the suggestion on putting together a book on soft errors and for initiating the related discussion with Springer staff, as well as for writing the foreword for this volume.

I would like also to thank Charles B. Glaser, Senior Editor and Ciara J. Vincent, Editorial Assistant at Springer, for their outstanding collaboration, as well as the Springer's production staff for final proofreading, editing, and producing the book.

My particular thanks to my colleague Raoul Velazco, for dedicating tremendous time in proofreading the text.

Grenoble, France — Michael Nicolaidis
February 2010

Contents

1 Soft Errors from Space to Ground: Historical Overview, Empirical Evidence, and Future Trends 1
Tino Heijmen

2 Single Event Effects: Mechanisms and Classification 27
Rémi Gaillard

3 JEDEC Standards on Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray Induced Soft Errors 55
Charles Slayman

4 Gate Level Modeling and Simulation 77
Nadine Buard and Lorena Anghel

5 Circuit and System Level Single-Event Effects Modeling and Simulation 103
Dan Alexandrescu

6 Hardware Fault Injection 141
Luis Entrena, Celia López Ongil, Mario García Valderas, Marta Portela García, and Michael Nicolaidis

7 Integrated Circuit Qualification for Space and Ground-Level Applications: Accelerated Tests and Error-Rate Predictions 167
Raoul Velazco, Gilles Foucard, and Paul Peronnard

8 Circuit-Level Soft-Error Mitigation 203
Michael Nicolaidis

9 Software-Level Soft-Error Mitigation Techniques 253
Maurizio Rebaudengo, Matteo Sonza Reorda, and Massimo Violante

10 Specification and Verification of Soft Error Performance in Reliable Electronic Systems 287
Allan L. Silburt, Adrian Evans, Ana Burghelea, Shi-Jie Wen, David Ward, Ron Norrish, Dean Hogle, and Ian Perriman

Index 313

Contributors

Dan Alexandrescu VP Engineering, Board Member, iRoC Technologies, WTC, PO Box 1510, 38025 Grenoble, France, dan@iroctech.com

Lorena Anghel Associate Professor, INPG TIMA Lab, 46 Avenue Félix Viallet, 38031 Grenoble, France, lorena.anghel@imag.fr

Nadine Buard Head of Electronic Systems Department, EADS Innovation Works, Suresnes, France, nadine.buard@eads.net

Ana Burghelea Cisco Systems, Advanced Manufacturing Technology Centre, 170 West Tasman Drive, San Jose, California 95134, USA

Luis Entrena Electronic Technology Department, Carlos III University of Madrid, Madrid, Spain, entrena@ing.uc3m.es

Adrian Evans Cisco Systems, SSE Silicon Group, 3000 Innovation Drive, Kanata, Ontario, K2K-3E8, Canada, adevans@sympatico.ca

Gilles Foucard TIMA Labs, 46 Avenue Félix Viallet, 38031 Grenoble, France, gilles.foucard@imag.fr

Rémi Gaillard Consultant on Radiation Effects and Mitigation Techniques on Electronics, Saint-Arnoult en Yvelines, France, remi.gaillard@iroctech.com

Marta Portela García Electronic Technology Department, Carlos III University of Madrid, Madrid, Spain, mportela@ing.uc3m.es

Tino Heijmen Regional Quality Center Europe, NXP Semiconductors, Nijmegen, The Netherlands, tino.heijmen@nxp.com

Dean Hogle Cisco Systems, SSE CRS Hardware, 3750 Cisco Way, San Jose, California 95134, USA, hogle@cisco.com

Michael Nicolaidis Research Director at the CNRS (French National Research Center), Leader ARIS group, TIMA Lab, av. Felix Viallet 46, 38031 Grenoble CX, France, michael.nicolaidis@imag.fr

Ron Norrish Cisco Systems, Technical Failure Analysis, 425 East Tasman Drive, San Jose, California 95134, USA, rnorrish@cisco.com

Celia López Ongil Electronic Technology Department, Carlos III University of Madrid, Madrid, Spain, celia@ing.uc3m.es

Paul Peronnard TIMA Labs, 46 Avenue Félix Viallet, 38031 Grenoble, France, paul.peronnard@imag.fr

Ian Perryman Ian Perryman & Associates, 160 Walden Drive, Kanata, Ontario, K2K-2K8, Canada, ian_perryman@primus.ca

Maurizio Rebaudengo Politecnico di Torino, Dip. di Automatica e Informatica, Corso Duca degli Abruzzi 24, 10129 Torino, Italy, maurizio.rebaudengo@polito.it

Matteo Sonza Reorda Politecnico di Torino, Dip. di Automatica e Informatica, Corso Duca degli Abruzzi 24, 10129 Torino, Italy, matteo.sonzareorda@polito.it

Allan L. Silburt Cisco Systems, SSE Silicon Group, 3000 Innovation Drive, Kanata, Ontario, K2K-3E8, Canada, asilburt@cisco.com

Charles Slayman Ops A La Carte, 990 Richard Ave. Suite 101, Santa Clara, CA 95050, USA, charlies@opsalacarte.com

Mario García Valderas Electronic Technology Department, Carlos III University of Madrid, Madrid, Spain, mgvalder@ing.uc3m.es

Raoul Velazco Director of Researches at CNRS (French Research Agency), TIMA Labs, 46 Avenue Félix Viallet, 38031 Grenoble, France, raoul.velazco@ imag.fr

Massimo Violante Politecnico di Torino, Dip. di Automatica e Informatica, Corso Duca degli Abruzzi 24, 10129 Torino, Italy, massimo.violante@polito.it

David Ward Juniper Networks, 1194 North Mathilda Avenue, Sunnyvale, California 94089, USA, dward@juniper.net

Shi-Jie Wen Cisco Systems, Advanced Manufacturing Technology Centre, 170 West Tasman Drive, San Jose, California 95134, USA, shwen@cisco.com

Chapter 1
Soft Errors from Space to Ground: Historical Overview, Empirical Evidence, and Future Trends

Tino Heijmen

Soft errors induced by radiation, which started as a rather exotic failure mechanism causing anomalies in satellite equipment, have become one of the most challenging issues that impact the reliability of modern electronic systems, also in ground-level applications. Many efforts have been spent in the last decades to measure, model, and mitigate radiation effects, applying numerous techniques approaching the problem at various abstraction levels. This chapter presents a historical overview of the soft-error subject and treats several "disaster stories" from the past. Furthermore, scaling trends are discussed for the most sensitive circuit types.

1.1 Introduction

Radiation-induced soft errors are an increasingly important threat to the reliability of integrated circuits (ICs) fabricated in advanced CMOS technologies. Soft errors are events in which data is corrupted, but the device itself is not permanently damaged. In contrast, a permanent device failure is called a hard error. Soft errors can have different effects on applications. On the one hand, they may result in data corruption at the system level, which may or may not be detected. On the other hand, soft errors can cause a malfunctioning of a circuit or even a system crash.

Soft errors are a subset of single-event effects (SEEs) and can be classified into the following categories [1] (see also Chaps. 2 and 3 of this book):

- *Single-bit upset* (*SBU*). A particle strike causes a bit-flip (upset) in a memory cell or a latch
- *Multiple-cell upset* (*MCU*). The event causes the upset of two or more memory cells or latches

T. Heijmen (✉)
Regional Quality Center Europe, NXP Semiconductors, Nijmegen, The Netherlands
e-mail: tino.heijmen@nxp.com

M. Nicolaidis (ed.), *Soft Errors in Modern Electronic Systems*,
Frontiers in Electronic Testing 41, DOI 10.1007/978-1-4419-6993-4_1,
© Springer Science+Business Media, LLC 2011

- *Multiple-bit upset* (*MBU*). The event causes the upset of two or more bits in the same word
- *Single-event transient* (*SET*). The event causes a voltage glitch in a circuit, which becomes a bit error when captured in a storage element
- *Single-event functional interrupt* (*SEFI*). The event causes loss of functionality due to the perturbation of control registers, clock signals, reset signals, lockup, etc.
- *Single-event latchup* (*SEL*). The event creates an abnormal high-current state by triggering a parasitic dual bipolar circuit, which requires a power reset. It can possibly cause permanent damage to the device, in which case the result is a hard error

The term SEU is also used, but unfortunately often in an ambiguous way. SEU is frequently applied as a synonym for soft error, but occasionally it is also used to describe all effects that are caused by a single strike of an energetic particle, including both soft and hard errors. Although strictly speaking it is not correct, the term "soft error" (or SEU) is often used to cover both SBUs and MBUs, which are the most common types of soft errors.

In terrestrial applications, soft errors are caused by either of two radiation sources:

- *Neutrons* generated by cosmic radiation interacting with the earth's atmosphere
- *Alpha particles* emitted by radioactive impurities that are present in low concentrations in chip and package materials

Before 1978, radiation was considered to be a reliability issue for space applications, but not for electronics operating at sea level. In space, radiation conditions are much more severe than on earth, in particular due to high-energy proton and heavy-ion rays. Under these conditions, not only soft errors occur, but also device degradation, especially if the total ionization dose (TID) is high. As will be discussed below, in 1978 it was demonstrated that radiation-induced soft errors are also present in electronic systems at sea level. TID effects, however, are unusual for terrestrial applications.

Usually, the soft-error rate (SER) is measured in FIT units (failures in time), where 1 FIT denotes one failure per billion device hours (i.e., one failure per 114,077 years). Typical SER values for electronic systems range between a few 100 and about 100,000 FIT (i.e., roughly one soft error per year).

In electronic components, the failure rate induced by soft errors can be relatively high compared to other reliability issues. Product monitoring shows that the hard-error failure rate, due to external events (such as electrical latchup), is maximally 10 FIT but usually much less. In contrast, the SER of 1 Mbit of SRAM, one of the most vulnerable types of circuits, is typically in the order of 1,000 FIT for modern process technologies. For a product that contains multiple Mbits of SRAM, the SER may be higher than the combined failure rate due to all other mechanisms. However, the effect of soft and hard errors is very different. In the case of a soft error, the product is not permanently damaged, and usually the error will disappear when

the corrupted data is overwritten. Furthermore, architectural and timing derating factors cause that the failure rate observed at the system level may be orders of magnitude lower than the combined SER of the memories in the product. Also, if a soft error occurs, in many cases it will manifest itself as a rather benign disturbance of the system without serious consequences. However, the occurrence of soft errors can have a serious effect on the perception that the customer has of the product's reliability.

The remainder of the current chapter is organized as follows: in Sect. 1.2, the history of soft errors is discussed, based on milestone reports. Section 1.3 treats the impact that soft errors can have on electronic systems, using some example "disaster stories." Finally, in Sect. 1.4, the scaling trends are discussed, with a focus on the most vulnerable circuit elements, i.e., volatile memories (SRAM and DRAM), sequential elements, and combinational logic.

1.2 History

In 1975, Binder et al. published the first report of soft errors in space applications [2]. The authors discussed four "anomalies" that had occurred in satellite electronics during an operational period of 17 years. According to their analysis, these four anomalies could not be attributed to the charging of the satellite by the solar wind. Instead, triggering of flip-flop circuits had caused the anomalies. The authors suggested that the failure mechanism was the charging of the base-emitter capacitance of critical transistors to the turn-voltage. The charges were produced by the dense ionization tracks of electron-hole pairs, generated by cosmic ray particles with high atomic numbers and high energy. Analysis showed that 100-MeV iron atoms could be responsible for producing the anomalies. It was assumed that this failure mechanism was not a serious problem because the corresponding failure rate was low (about one fail in 4 years). Heavy ion rays, such as the 100-MeV iron particles, are not capable of crossing the earth's atmosphere. Therefore, these radiation-induced anomalies were assumed to be absent in terrestrial electronics.

In 1978, May and Woods of Intel presented a paper at the International Reliability Physics Symposium (IRPS) on a new physical mechanism for soft errors in DRAMs [3]. This publication introduced the definition of "soft errors" as random, nonrecurring, single-bit errors in memory elements, not caused by electrical noise or electromagnetic interference but by radiation. The paper reported on soft errors in the Intel 2107-series 16-kb DRAMs, which were caused by alpha particles emitted in the radioactive decay of uranium and thorium impurities in the package materials. It was the first public account of radiation-induced soft errors in electronic devices at sea level.

The authors discussed that electron-hole pairs are generated when alpha particles interact with silicon. Depletion layers can collect these charges, and the generated electrons can end up in the storage wells of the memory elements. If the amount of collected charge exceeds a critical value Q_{crit}, a soft error occurs, see Fig. 1.1.

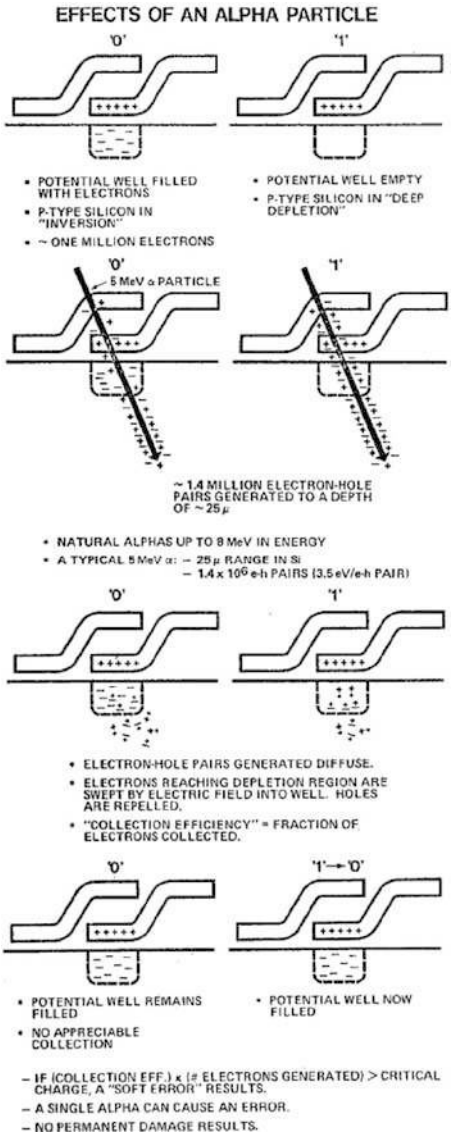

EFFECTS OF AN ALPHA PARTICLE
'0'
'1'
• POTENTIAL WELL FILLED WITH ELECTRONS
• P-TYPE SILICON IN "INVERSION"
• ~ ONE MILLION ELECTRONS
• POTENTIAL WELL EMPTY
• P-TYPE SILICON IN "DEEP DEPLETION"
'0'
5 MeV α PARTICLE
'1'
~ 1.4 MILLION ELECTRON-HOLE PAIRS GENERATED TO A DEPTH OF ~ 25 μ
• NATURAL ALPHAS UP TO 8 MeV IN ENERGY
• A TYPICAL 5 MeV α: – 25 μ RANGE IN Si
– 1.4 x 10^6 e-h PAIRS (3.5 eV/e-h PAIR)
'0'
'1'
• ELECTRON-HOLE PAIRS GENERATED DIFFUSE.
• ELECTRONS REACHING DEPLETION REGION ARE SWEPT BY ELECTRIC FIELD INTO WELL. HOLES ARE REPELLED.
• "COLLECTION EFFICIENCY" = FRACTION OF ELECTRONS COLLECTED.
'0'
'1'→'0'
• POTENTIAL WELL REMAINS FILLED
• NO APPRECIABLE COLLECTION
• POTENTIAL WELL NOW FILLED
– IF (COLLECTION EFF.) x (# ELECTRONS GENERATED) > CRITICAL CHARGE, A "SOFT ERROR" RESULTS.
– A SINGLE ALPHA CAN CAUSE AN ERROR.
– NO PERMANENT DAMAGE RESULTS.

Fig. 1.1 Alpha particles creating soft errors in DRAMs (figure from [3])

The story of the SER problem in the Intel 2107-series has been discussed in detail in the well-known special issue of *IBM Journal of Research and Development* of January 1996 [4]. Because it was an important industrial problem for Intel, investigations were continued until the root cause was discovered. It was found that the ceramic package of the product was contaminated with radioactive impurities from the water that was used in the manufacturing process. The package factory had

been built along a river, downstream from an old uranium mine. Waste from the mine had contaminated the water and, indirectly, the packages.

The paper by May and Woods has become famous and started a tradition of research on soft errors in sea-level applications. The preprint of the paper circulated in the reliability community of the semiconductor industry and even newspapers spent articles on this subject. Three decades later, an increasing number of researchers and engineers are working on soft errors, and numerous papers on this subject are published than ever before.

In 1978, Ziegler of IBM had the idea that if alpha particles can induce soft errors, possibly also cosmic radiation may have the same effect [5]. In particular, cosmic ray particles might interact with chip materials and cause the fragmentation of silicon nuclei. The fragments could induce a local burst of electronic charges, resulting in a soft error. The physical mechanisms behind these events are described in Chap. 2 of this book. Ziegler worked on this problem together with Lanford of Yale University for about a year. They found that cosmic particles in the solar wind were not able to penetrate the earth's atmosphere. Only intergalactic particles with energies of more than 2 GeV can cause soft errors in sea-level electronics, albeit in an indirect way. The high-energy cosmic rays interact with particles in the atmosphere and cause a cascade of nuclear reactions, see Fig. 1.2. It is only the sixth generation of particles that will reach sea level. This generation consists of neutrons, protons, electrons, and transient particles such as muons and pions. Ziegler and Lanford showed how these particles interact with silicon, combining the

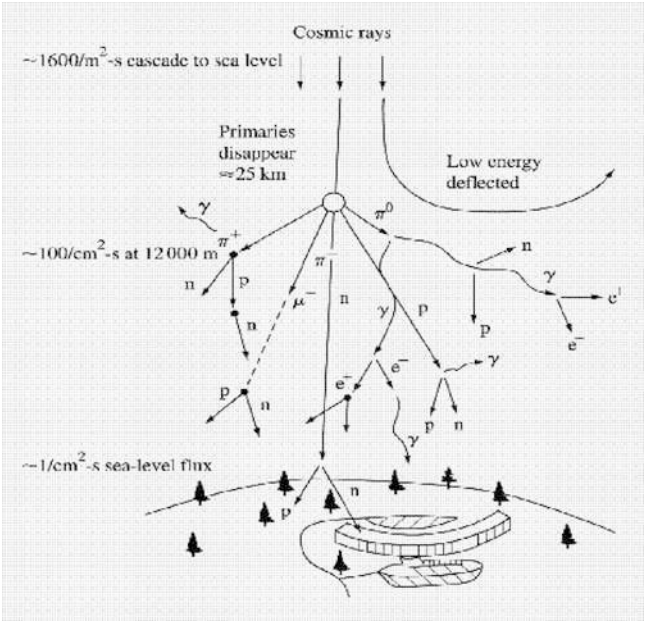


Fig. 1.2 Schematic view of cosmic rays causing cascades of particles (figure from [4])

particle flux with the probability that such a particle can cause a sufficiently large burst of charge.

This paper was followed by a more detailed study that also addressed the amount of charge that was needed to upset a circuit [6]. Because of the usage of materials with low alpha emission rates, cosmic neutrons replaced alpha particles as the main source of memory SER during the 1990s. However, due to the reduction of critical charges, the SER contribution from alpha particles has gained importance again during the last years.

IBM started unaccelerated real-time SER tests in 1983, using a portable tester with several hundreds of chips. Real-time SER testing is also named field-testing or system SER (SSER) testing. The IBM measurements provided evidence that even at sea level cosmic radiation contributes significantly to the SER, and its effect increases exponentially with altitude [7, 8]. It was also shown that there is a close correlation between the SER and the neutron flux. The test results further indicated that cosmic rays generate a relatively large number of multiple bit errors, because the generated charge is in general much larger than for an alpha-particle event.

Lage et al. of Motorola provided further evidence that the SER of circuits is not exclusively caused by alpha particles [9]. The authors collected data for various SRAM types, obtained from real-time SER measurements in which a large number of memory devices were tested for a long time and the occurring soft errors were logged. These data were compared with the corresponding accelerated SER measurements, where memories were exposed to a high flux of alpha particles. If alpha particles were the only source of soft errors, the correlation between the two sets of experimental data would be linear. However, when the real-time SER results are plotted against data from accelerated measurements, as shown in Fig. 1.3, there is a clear discrepancy from the linear correlation. The real-time SER is larger than that would be expected from the results of the accelerated tests with alpha sources, in particular for low SER values. The difference can be explained if the contribution from neutrons is included.

In 1995, Baumann et al. from Texas Instruments presented a study that showed that boron compounds are a nonnegligible source of soft errors [10]. Two isotopes of boron exist, ^{10}B (19.1% abundance) and ^{11}B (80.1% abundance). Different from other isotopes ^{10}B is highly unstable when exposed to neutrons. Furthermore, while other isotopes emit only gamma photons after absorbing a neutron, the ^{10}B nucleus fissions (i.e., breaks apart), producing an excited ^{7}Li recoil nucleus and an alpha particle. Both particles generate charges in silicon and can therefore cause soft errors. Although neutrons with any energy can induce fission, the probability decreases rapidly with increasing neutron energy. Therefore, only thermal (i.e., low-energy) neutrons need to be considered. It has been shown that neutrons with energies below 15 eV cause 90% of the reactions [11]. Because thermal neutrons are easily scattered, the local environment has a large influence on the flux. Therefore, the background flux for low-energy neutrons is not well defined.

Boron is used extensively both as a p-type dopant and in boron phosphosilicate glass (BPSG) layers. Boron is added to PSG to improve the step coverage and contact reflow at lower processing temperatures. Typically, the ^{10}B concentration in

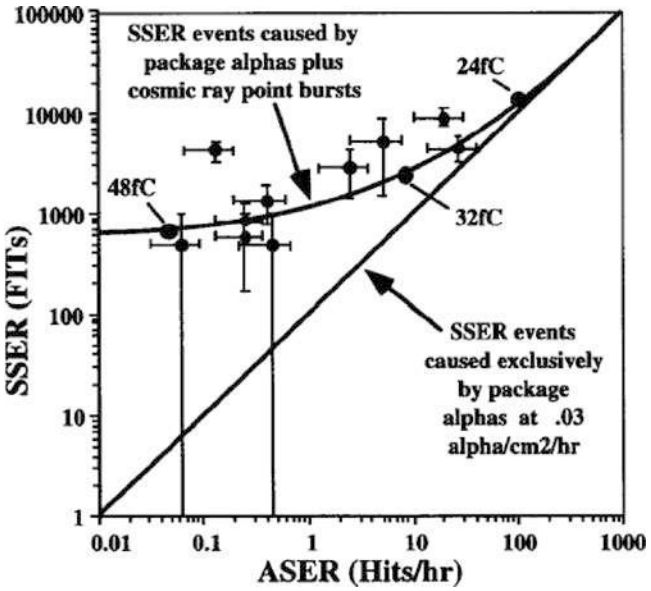


Fig. 1.3 Correlation between data from unaccelerated real-time SER measurements, denoted system SER (SSER), and alpha-accelerated SER measurements on SRAMs (figure from [9])

BPSG is thousands of times higher than in diffusion and implant layers. For conventional Al-based processes, BPSG is the dominant source of boron fission and, in some cases, the primary source of soft errors [12]. Only BPSG in close proximity to the silicon substrate is a threat because the range of both the alpha particle and the lithium recoil is less than 3 μm. In most cases, the emitted particles have insufficient energy beyond 0.5 μm to create a soft error. BPSG is generally applied in process technologies using aluminum backend. In copper-based technologies, metal layers are processed in a different manner, using chemical-mechanical polishing, which does not require the specific properties of BPSG. Because of this, thermal neutron-induced boron fission is not a major source of soft errors in advanced CMOS technologies using copper interconnect.

1.3 Soft Errors in Electronic Systems

Whether or not soft errors impose a reliability risk for electronic systems strongly depends on the application. Soft-error rate is generally not an issue for single-user consumer applications such as mobile phones. However, it can be a problem for applications that either contain huge amounts of memories, or have very severe reliability requirements.

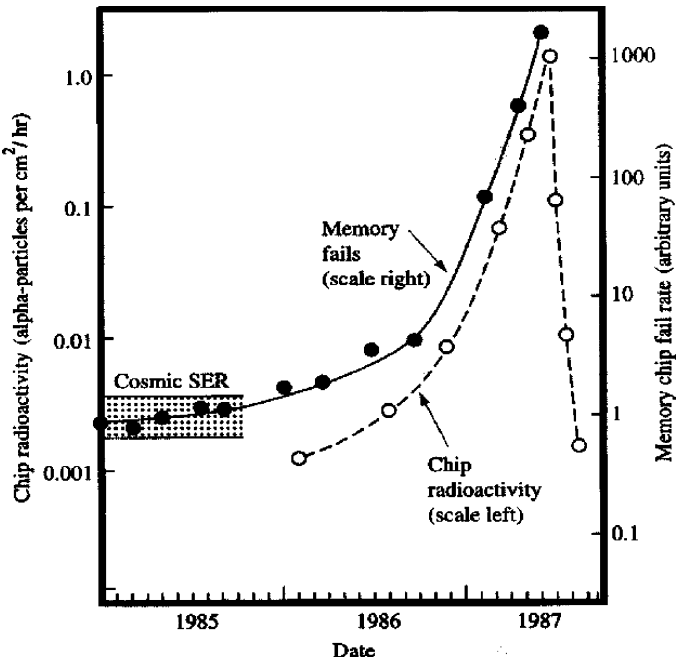


Fig. 1.4 Chip radioactivity and memory failure rate of the IBM LSI memory during the "Hera" problem (figure from [4])

If the effect of soft errors is manifested at the system level, it is generally in the form of a sudden malfunctioning of the electronic equipment, which cannot be readily attributed to a specific cause. Soft errors are untraceable once new data have been written into the memory that stored the corrupted bits or when the power of the device has been reset. Therefore, failure analysis is not capable of identifying soft errors as the root cause of the problem. Furthermore, the problem is not reproducible, due to its stochastic nature. Because of this, it is usually very difficult to show that soft errors are causing the observed failures. In the semiconductor industry, several examples are known that confirm this. In the case of the 2107-series DRAM of Intel, discussed in Sect. 1.2, it took many efforts to find out that the water used in the package factory was causing the contamination with radioactive impurities and that this was the root cause of the problem.

An even more spectacular example is what is now known as the "Hera" problem of IBM [4]. During 1986, IBM observed an increase in failures of their LSI memories manufactured in the USA. Surprisingly, identical memories that were produced in Europe did not show this problem. Knowing the case of the 2107-series DRAM of Intel, the ceramic package was identified as a possible cause of the fails. Therefore, the U.S. chips were assembled in European packages and vice versa. It was found that the U.S. chips (in the European packages) gave a high failure rate, whereas the European chips (with the U.S. packages) did not show failures.

This clearly demonstrated that the problem was not in the package but in the memory die.

While the problem was becoming very serious from a business point of view, further analysis showed that the chips had a significantly high radioactivity. The problem was then to find the root cause. Traces of radioactivity were found in different processing units, and it appeared that ^{210}Po was the radioactive contaminant. Surprisingly, the investigators found that the chip radioactivity that appeared in 1986 had increased up to a factor of 1,000 in May 22, 1987 and then disappeared (see Fig. 1.4)! After this discovery, it took months of investigation to identify a bottle of nitric acid that was used in the wafer processing as a contamination source. At the supplier's factory, it was found that a machine that was used to clean the bottles caused the radioactivity. This machine used an air jet that was ionized with ^{210}Po to remove electrostatic dust inside the bottles after washing. The radioactive ^{210}Po in the air jet capsule were sealed. Because the supplier of the equipment had changed the epoxy that was used for this seal, the jets were occasionally and randomly leaking radioactivity. As a result, a few out of thousands of acid bottles were contaminated. Once the root cause was identified, all contaminated acid bottles were removed and the problem completely disappeared.

In the end, it appeared that what seems a rather trivial issue with a bottle-cleaning machine had caused a serious problem for IBM that had lasted for more than a year. Many man-hours had been spent to solve the issue, and the problem had affected IBM's business.

The two examples discussed above were caused by a sudden contamination with radioactive impurities. However, also if the background radiation does not exceed the usual level, soft errors can cause serious problems in electronic equipment. In the last decade, this has been demonstrated by two major issues related to radiation-induced soft errors.

The first case is the problem in the high-end server line "Enterprise" of Sun in 1999/2000. This problem has been reported in a legendary article in Forbes magazine [13]. The Enterprise was the flagship in Sun's server line and was used by many companies, including major ones such as America Online and Ebay. The cost price ranged from $50,000 to more than $1 million. During 1999, some of the customers reported that occasionally the server crashed for no apparent reason. For a Web-based company, which is supposed to be online 24 h/d, this is a serious problem. One company reported that their server had crashed and rebooted four times within a few months.

It took Sun months of tedious investigations to identify that soft errors in the cache memory of the server were the root cause. Until then, long tests had been run without problems on machines that had crashed in the field. The cache modules of these machines contain SRAM chips vulnerable to soft errors. Over the years, from generation to generation the number of SRAM chips per server and the bit count per SRAM chip had increased. Furthermore, the soft-error vulnerability of the SRAMs had become worse, due to technology scaling. As a result, the SER of the complete system increased from one product generation to the next, until the point where soft errors became the dominant source of system failures and the mean time between

failures (MTBF) became unacceptably high. Because of this, other suppliers protected their cache memories with error-correction coding, but Sun had simply missed it.

When the root cause had been identified, the caches in the server were replaced by "mirrored" ones, where if the first cache fails, the second one is a back up. Sun stated that the issue had cost tens of millions of dollars and a huge amount of man-hours, both at the technical level and in briefings with customers. Furthermore, Sun's brand image was damaged and the company was criticized for the treatment of their customers. The statement by one of their customers: "It's ridiculous. I've got a $300,000 server that doesn't work. The thing should be bulletproof!" has become a famous one-liner in the industry. From a technical point of view, this case demonstrated that soft-errors could cause problems for systems that contain large amounts of memory and that the expected SER should be analyzed carefully during the development stage.

Another company that encountered a major SER issue was Cisco systems in 2003 [14]. Router line cards of the 12000 series, with a selling price of about $200,000, showed failures caused by radiation-induced soft errors. Parity errors in the memories and ASICs resulted in a reset, during which the card was reloaded. This reloading took 2–3 min of recovery time. After the card reloaded, data was passing normally. The problem was solved by a new release of the Internetwork operating system (IOS) software, which included several improvements for error recovery. These improvements reduced the probability that a card has to reload due to a soft error, reduced the reload time if necessary, and provided better text messaging about the failures.

1.4 Scaling Trends

At the time when SER was discovered as a significant reliability issue for terrestrial applications, DRAMs were the most vulnerable circuit elements. SRAMs were more robust then because pull-up and pull-down transistors stabilize the charges representing the memory state. However, due to major technology changes, DRAMs have become more robust against soft errors with every generation. In contrast, SRAMs have become more vulnerable with technology scaling. This is mainly caused by the downscaling of the supply voltage and by the reduction of the minimum feature sizes. Because of these trends the number of electrons that represent a bit in an SRAM bit cell has decreased from about one million to a few thousands.

Since a few technology generations, however, the SER vulnerability of an SRAM bit cell has saturated or is even decreasing. This is because not only the so-called critical charge Q_{crit} that must be injected in order to generate a soft error has decreased, but also the silicon volume from which induced charges can be collected. The latter trend is dominating over the former one for the most advanced

CMOS nodes, resulting in a decrease in the SER per bit. An important factor is that the reduction of the supply voltage is much less aggressive than before.

In the current section, the SER scaling trends for different types of circuits (SRAM, DRAM, sequential logic, and combinational logic) are discussed.

1.4.1 Scaling of SRAM SER

Early SRAM was significantly more robust against radiation-induced soft errors than DRAM. This was because in an SRAM bit cell the data is stored in a feedback loop of two cross-coupled inverters. This feedback loop is forcing the bit cell to stay in its programmed state. However, with technology scaling the supply voltage and the node capacitance decreased, which resulted in a lower critical charge Q_{crit} with every SRAM generation.

Cohen et al. published one of the first SER trends [15]. They determined experimentally the alpha-particle-induced SER of both SRAM and dynamic logic arrays. The authors found that the SER decreased exponentially with decreasing supply voltage at 2.1–2.2 decades/V. Based on the SIA roadmap projections for operating voltages, they predicted the scaling trend for the FIT/Mbit, see Fig. 1.5. The error bars account for the range of supply voltages reported by the SIA roadmap. The impact of the reduction in feature sizes was not taken into account.

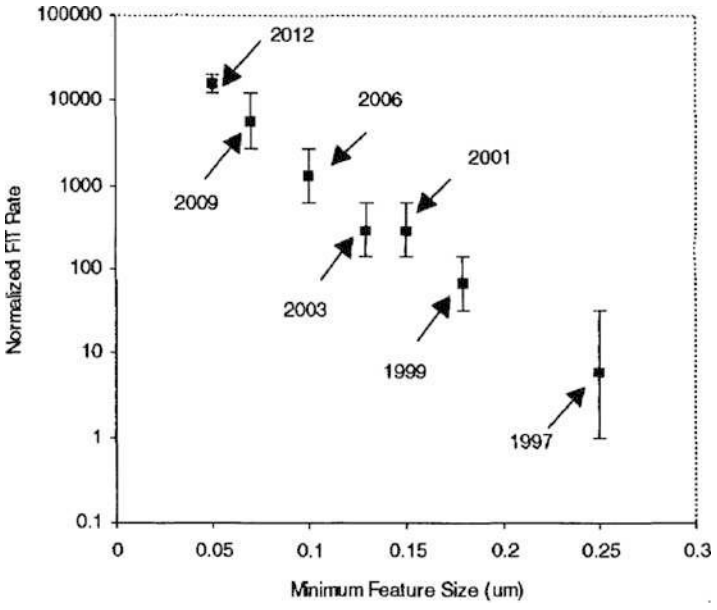


Fig. 1.5 Scaling trend for the FIT/Mbit of SRAM and dynamic logic arrays, predicted in 1999 (figure from [15])

The authors estimated that the FIT/Mbit would increase with nearly 100× within one decade.

When feature sizes were scaled down further and further into the deep submicron regime, the SRAM SER/bit trend started to saturate. This was mainly caused by the fact that the downscaling of the supply voltage was saturating. A typical SRAM SER per bit scaling trend is shown in Fig. 1.6, reproduced from a Baumann's publication [16]. For the processes applied by TI, it was found that a major part of the soft errors were caused by low-energy cosmic neutrons interacting with ^{10}B in the BPSG. When this material was removed from the process flow, the SRAM SER per bit showed a significant decrease. The dotted line in Fig. 1.6 represents the SRAM SER per bit trend assuming that BPSG has not been removed.

For the most advanced technology nodes, the SRAM SER per bit shows a decrease. An example is shown in Fig. 1.7, where the SER per bit of SRAM caches is depicted, as published by Seifert et al. of Intel [17]. The trend shows a peak at the 130-nm node and is decreasing since then. This decrease is caused by the fact that the average collected charge is downscaling in the same pace as the critical charge. Because the ratio of these two parameters is fairly constant, it is the reduction in vulnerable junction diffusion area that is driving the decrease in SRAM SER per bit. Due to this decrease in vulnerable area, the probability that an SRAM bit cell is hit by a particle is lower. Although the SRAM SER is dependent on the details of the process technology, the common trend in the industry is that the SRAM SER per bit peaks at 130–90 nm and shows a decrease after that node.

SRAMs manufactured in silicon-on-insulator (SOI) technologies generally have a lower SER than their counterparts processed in bulk CMOS in the same technology node. This is because the presence of a buried oxide (BOX) layer reduces the sensitive volume of a transistor in SOI. A data analysis by Roche and Gasiot of STMicroelectronics showed that partially depleted (PD) SOI has a 2–6× lower SER than bulk CMOS [18]. Recent work of Cannon et al. of IBM showed that SRAMs processed in 65-nm SOI technology have a 3–5× lower SER than SRAMs

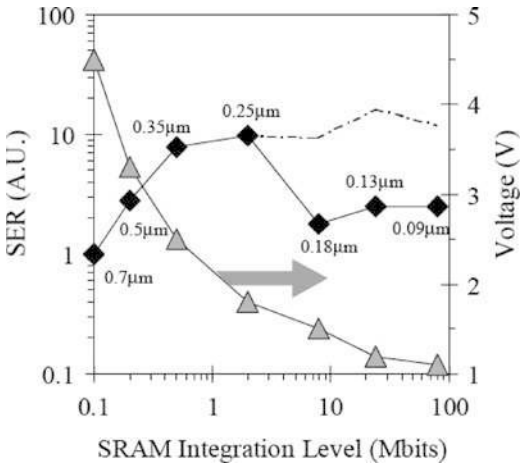


Fig. 1.6 SRAM SER per bit as a function of technology node (*black diamonds*). The dotted line represents the scaling trend if BPSG would not have been removed from the process flow. The *gray triangles* show the downscaling of the supply voltage (figure from [16])

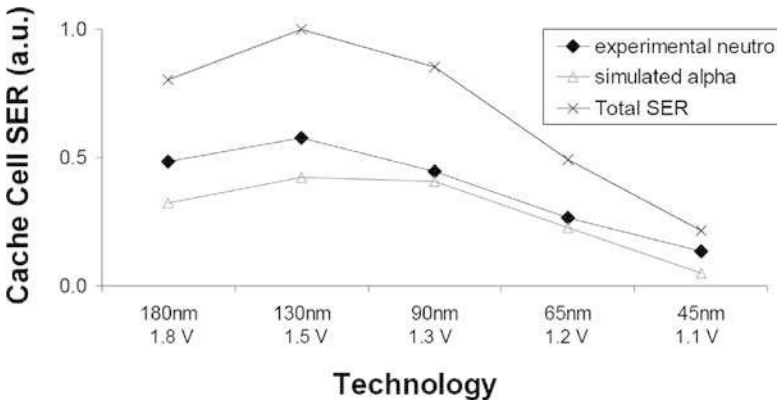


Fig. 1.7 Normalized SER per bit trend for SRAM caches of Intel products (updated figure from [17])

manufactured in 65-nm bulk CMOS, in agreement with results from previous technology nodes [19]. The SER vulnerability in fully depleted (FD) SOI is even lower than in PD SOI.

SRAMs manufactured in advanced technologies show a spread in SER caused by variations in the process parameters. Experimental results of Heijmen and Kruseman of Philips demonstrated that if two chip designs contain embedded SRAM instances of the same type, then the SER can be different with as much as 40% [20]. Additionally, significant batch-to-batch and sample-to-sample variations in SER have been observed. Furthermore, process variability causes that the SER vulnerability of an SRAM bit cell is not the same for its two data states. Experimental results for a 90-nm embedded SRAM showed that the differences can be almost a factor of 4 [21]. In general, however, this data dependency is hidden because it is common that the mapping of logical on physical states in memories is interchanged column-by-column. That is, if a logical "1" corresponds to one physical state in one column, it corresponds to the complementary state in the next column.

Ongoing reduction of feature sizes has increased the probability that a single particle causes an MCU. The percentage of MCU events is rapidly increasing with technology scaling and can be several tens of percent in 65- and 45-nm processes. Two mechanisms are known that can produce MCUs. On the one hand, radiation-induced charges can be shared by neighboring bit cells. On the other hand, the injected charge can trigger a parasitic bipolar transistor, which results in an increase in the amount of charge collected by these neighboring bit cells. In the case of bipolar amplification, also known as the "battery effect" [22], the MCU rate increases with increasing supply voltage, whereas in the case of charge sharing the MCU rate decreases if the voltage is raised. In general, the relative MCU rate is higher for neutrons than for alpha particles because a neutron generates more charges in silicon and also because a neutron hitting an atom in a semiconductor device often produces multiple ionizing particles. It has been observed that a single

neutron caused more than 50 bit flips in an SRAM. During neutron-accelerated SER tests of SRAMs in advanced processes, it is not unusual to observe that more than 50% of the events are MCUs. In contrast, during alpha-accelerated SER tests typically the majority of the observed events are SBUs, with only a few percent MCUs.

An MCU is called MBU if the affected bits are in the same word. The occurrence of MBUs affects the error-correction coding (ECC) that is used to protect the SRAM. The most generally applied ECC techniques use single-error correct, double-error detect (SEC-DED) schemes, which implies that they are not capable of correcting MBUs. Physical interleaving strongly reduces the MBU rate. With interleaving, also named scrambling, bits from the same logical word are physically separated over the memory. As a result, in the case of an MCU the corrupted bits generally belong to different words. Because of this hampering of the protection with ECC, many research projects in the field of radiation effects currently focus on MCUs. An overview of ECC approaches can be found in Chap. 8 of this book.

1.4.2 Scaling of DRAM SER

Although in the first report by May and Woods soft errors in dynamic memories were caused by alpha particles [3], in deep submicron CMOS process technologies neutrons dominate over alpha particles as the main cause of soft errors in dynamic random access memories (DRAMs). As process technology scales down below 100-nm, SETs in the address and control logic of the memories are becoming more important, compared to the upsets of the memory bit cells. This is because the capacitance of the bit cells is not decreasing with technology scaling. Whereas initially planar 2D capacitors with large junction areas were used, 3D capacitors were developed. These were not only beneficial for SER, but also improved the refresh performance of the DRAMs. Because the cell capacitance is hardly affected by technology scaling, the critical charge of a bit cell remains roughly constant also. Another advantage of the introduction of 3D capacitance was that the charge collection efficiency was significantly reduced because the vulnerable junction volume is much smaller. With technology scaling, this junction volume is decreasing further, which results in a lower SER per bit. In contrast, address and control logic circuitry in the periphery of the memory is becoming more susceptible to soft errors because their node capacitances are decreasing with technology scaling. These so-called logic upsets result in massive amounts of bit errors (typically several thousands) because a wrong part of the memory is addressed. The rate of logic errors per bit is not scaling down with DRAM process technology.

The cell-upset rate of DRAMs strongly depends on the type of technology that is applied for the bit cell. In a 1998 paper Ziegler and coauthors investigated the SER of DRAMs with stacked-capacitor (SC) cells, trench cells with external charge (TEC cells), and trench cells with internal charge (TIC cells) [23]. The cosmic-ray-induced SER in 26 different chips with 16 Mbit of DRAM was evaluated. The SER

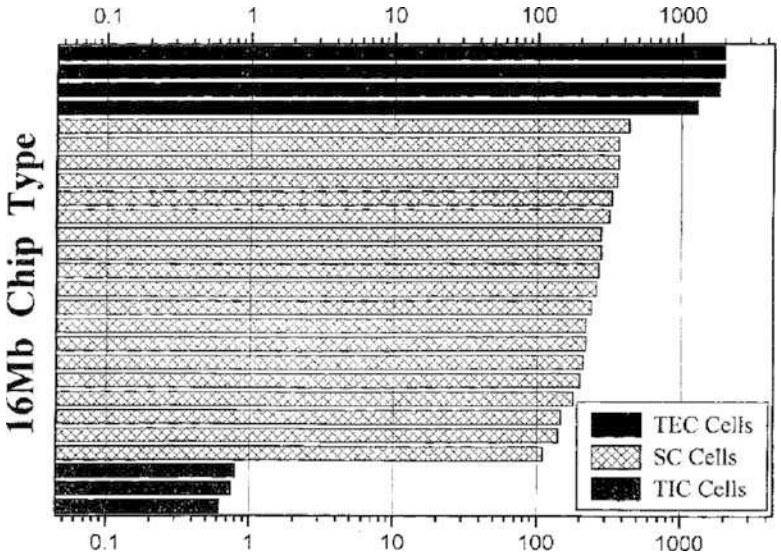


Fig. 1.8 Correlation between cosmic-ray-induced DRAM SER and bit cell technology: TEC (*up*), SC (*middle*), and TIC (*low*) (figure from [23])

of the TEC cells appeared to be about 1,500× higher than the SER of the TIC cells, with the SC cells having an intermediate SER, see Fig. 1.8.

In a dynamic memory, only the charged bit cells are susceptible to upsets, cf., Fig. 1.1. However, in most DRAMs the SER is independent of the stored data pattern. This is because a logical "1" ("0") does not mean that the data is stored as a charged (uncharged) bit cell. Instead, different translation schemes from the logical to the physical level are generally used in different parts of the memory. Therefore, the probability that a "1" is stored as a charged cell is typically equal to the probability that it is stored as an uncharged cell. The cell upset rate is typically independent of the frequency of the memory. However, in a recent publication Borocki et al. of Infineon observed a decreasing trend with frequency for the logic upset rate [24].

At the system level, the bit error rate (BER) is what is observed. Therefore, the number of events is less important than the number of detected errors, i.e., the effective BER at the system level. Whereas the event rate per bit is decreasing with technology scaling, there is not a clear trend for the BER, see Fig. 1.9 [24]. The trend is that DRAM chips with an increasing density are used and that more DRAM chips are applied in a system. Because of this, the BER at the system level will increase. Also, the logic error rate of a system is growing. Therefore, the importance of using error-correcting coding (ECC) to detect and correct soft errors in DRAMs is increasing for the applications that require a high reliability. Because logic errors often result in a multiple-bit error observed by the system, it will be necessary to develop error correction codes that are more advanced than single-error correct, double-error detect (SEC–DED) schemes.

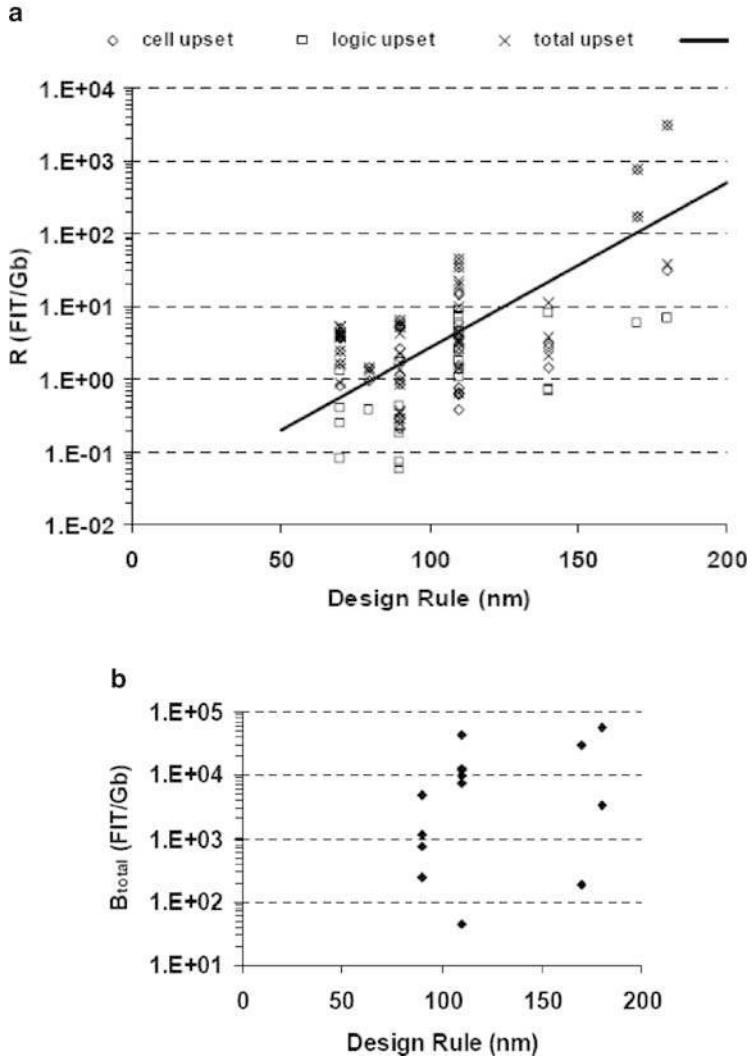


Fig. 1.9 DRAM *event* (**a**) and *bit error* (**b**) rate. Data have been normalized to the neutron flux at New York City for a 1-Gb chip (figures from [24])

1.4.3 SER of Latches and Flip-Flops

Sequential elements, such as latches and flip-flops, are digital logic circuits that are used for temporary data storage. This type of circuitry is also denoted sequential logic because the output depends not only on the input, but also on the history of the input. A flip-flop basically contains two latches connected in series. Similar to SRAMs the memory function in sequential elements is based on a feedback loop

formed by two cross-coupled inverters. However, with respect to SER vulnerability, there are some essential differences between SRAM, on the one hand, and latches and flip-flops, on the other hand:

- In general, SRAM bit cells are symmetric by design and latches are not. As a result, the SER of a latch depends on the data that is stored. In contrast, the SER of an SRAM bit cell is the same when a "1" or a "0" is stored, except for the impact of process variability, as discussed in Sect. 1.4.1
- A latch is vulnerable to upsets only if its internal clock state is such that the latch is storing data. In the complementary clock state, when the latch is transparent, no bit flip can be generated because no data is stored. For example, in a D-type flip-flop, the master latch is vulnerable to upsets for one clock state and the slave latch for the complementary state. In contrast, SRAM bit cells are not connected to clock signals, and therefore SRAM SER is not dependent on the clock state
- Most SRAMs use 6-transistor (6-T) bit cells. These bit cells are designed such that their area is minimal, with additional constraints especially on speed and power dissipation. For a given process technology in general, only a limited set of SRAM bit cells are available, e.g., a high-density, a high-performance, and a low-power variant, with relatively small differences in cell layout. In contrast, dozens of design styles are used to construct sequential elements. Area is much less a concern here because in general the number of latches and/or flip-flops in an IC is orders of magnitude less than the amount of SRAM bit cells. As a result, latch and flip-flop SER shows a much larger variation between different cell designs

Heijmen et al. of Philips, STMicroelectronics, and Freescale Semiconductor published measured alpha- and neutron-SER data of five different flip-flops from a standard-cell library [25]. The flip-flop cells differed in threshold voltage, circuit schematic, cell height, drive strength, and/or functionality. The results, illustrating the dependencies of the flip-flop SER on data state, clock state, and cell type, are shown in Fig. 1.10.

The anomalies in satellite electronics, reported by Binder et al. [2] in 1975, were due to upsets of flip-flops. However, in sea-level applications, the SER contribution from sequential logic used to be negligible in older process technologies, compared to SRAM and DRAM. This was because logic circuits are generally much larger than memory bit cells and therefore have relatively large critical charges, which made logic circuitry virtually immune to soft errors in old processes. But with technology scaling Q_{crit} reduced and as a result, the SER vulnerability of latches and flip-flops increased. At about the 0.13 μm technology node, the FIT/Mbit of sequential elements became large enough to contribute substantially to the chip-level SER, depending on the amount of cells. Some scaling trends for the average latch and flip-flop SER, reported by Seifert et al. of Intel [17], Baumann of TI [26], and Heijmen and Ngan of NXP Semiconductors [27] are shown in Fig. 1.11. Most scaling trends for the SER of latches and of flip-flops show saturation, followed by a decrease in the SER/bit beyond the 90-nm. In modern CMOS processes, the average SER/bit of a flip-flop/latch is comparable to the one of SRAM.

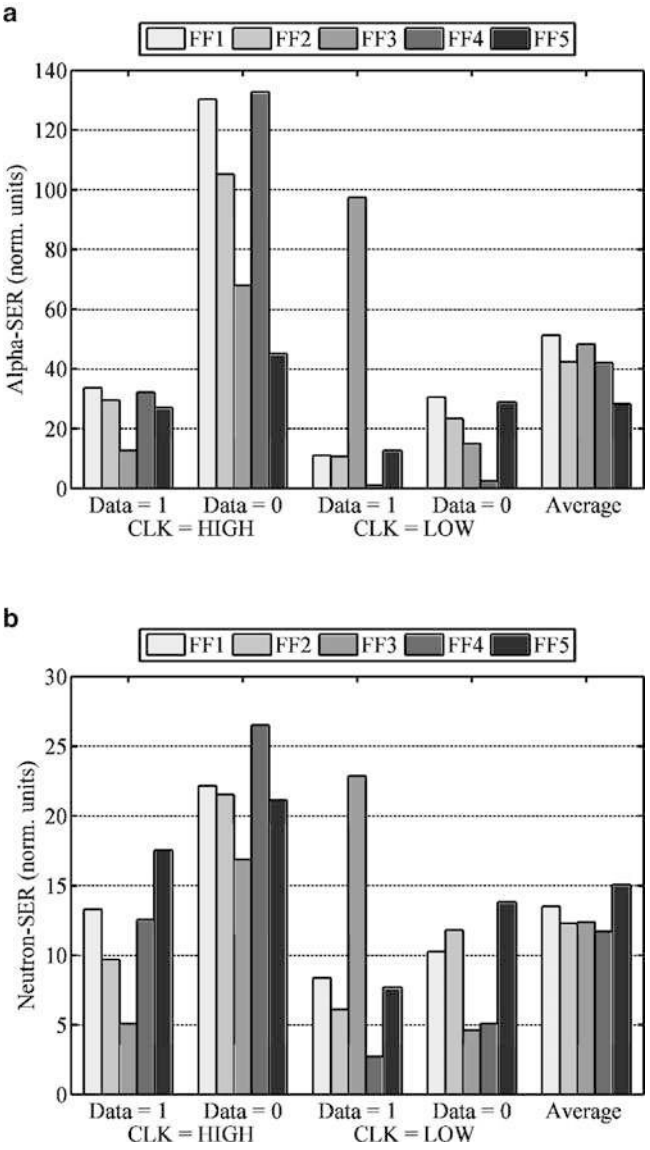


Fig. 1.10 Alpha-SER (**a**) and neutron-SER (**b**) of five different flip-flops from a 90-nm library as a function of clock and data state (figures from [25])

Different from memories, logic circuits cannot be efficiently protected against soft errors with the use of ECC, due to the lack of a regular structure. Instead, many latches and flip-flops with an improved SER performance have been published in the literature. However, these solutions often suffer from severe penalties in terms of area, power dissipation, and speed. Often radiation-hardened latches apply

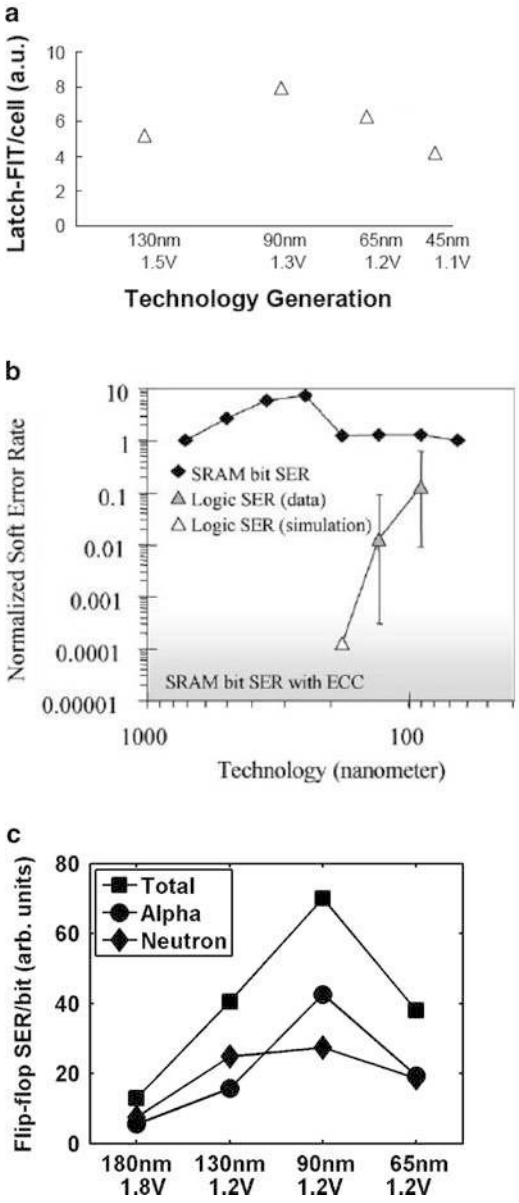


Fig. 1.11 Scaling trends for (**a**) latches [17]; (**b**) flip-flops and latches [26]; and (**c**) flip-flops [27]

redundancy in order to reduce the SER vulnerability. As a result, such cells generally have a large amount of internal interconnect, which makes that in many cases the lower metal layers are blocked for routing. Furthermore, the effectiveness of redundancy-based radiation-hardened latches is decreasing with technology

scaling, as charge sharing and upsets at internal clock nodes become more important, as discussed by Seifert et al. of Intel [28].

1.4.4 SER of Combinational Logic

Soft errors in combinational logic are generated by a different mechanism than in memories or logic storage cells. First, an ionizing particle causes an SET, i.e., a voltage glitch. The SET propagates through the circuit and results in a soft error if it is captured by a storage element. Three masking effects reduce the probability that an SET is captured:

- *Timing* masking: the SET arrives too soon or too late at the input of the storage element to be captured.
- *Electrical* masking: the pulse is attenuated during its propagation such that its amplitude is too small for the SET to be captured.
- *Logical* masking: the inputs of the combinational circuit are such that the logical path for the propagation of the SET is blocked.

Another difference with SER in memories and flip-flops/latches is that SER in combinational logic is increasing with the operational frequency. This is because the probability that an SET is captured increases with the number of capturing moments per time unit. Furthermore, the design of the circuit and the algorithm that is performed both strongly impact the probability that a SET in the combinational logic will affect the operation of the system.

It has been calculated (Mitra et al. of Intel [29]) that for high-frequency applications, such as microprocessors, network processor, and network storage controllers implemented in modern processes, about 10% of the soft errors originate in the combinational logic, see Fig. 1.12. Typically, in such applications, the larger memory modules are protected with ECC. The smaller memories are not protected

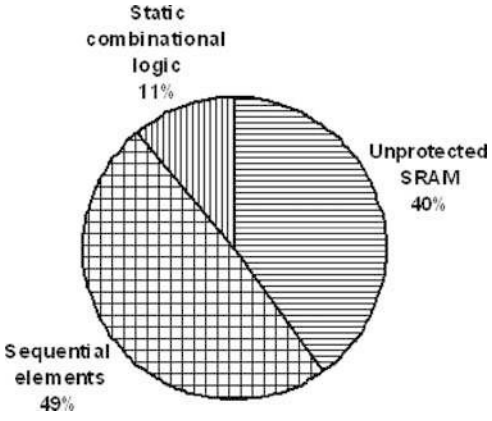


Fig. 1.12 Contributions to chip-level SER for applications such as microprocessors (recreated figure from [29])

because the penalties for using ECC are unacceptably high for such small blocks. These unprotected memory modules contribute to the chip-level SER. In a recent publication, Gill et al. from Intel showed that at the 32-nm node SER in combinational logic is not a dominant contributor at the chip-level [30].

1.4.5 Scaling of Single-Event Latchup

Alpha particles and cosmic neutrons cannot only generate soft errors, but also cause SEL. The mechanism that causes SEL is similar to the mechanism for electrical latchup, the difference being that SEL is triggered by radiation-induced charges. Because a CMOS process contains both NMOS and PMOS transistors, parasitic thyristors are present, which can be switched on if their voltage levels meet certain conditions. The consequences of an SEL are often more severe than that of a soft error because a power-reset is required, and the IC can be permanently damaged if it is not protected with current delimiters. Also, SEL can impact the long-term reliability of the IC.

SEL rates exceeding 500 FIT/Mbit have been reported by Dodd et al. of Sandia for SRAMs processed in a 0.25 μm technology [31]. The voltage and temperature dependencies of the SEL rate are different than for SER. This is because the underlying upset mechanisms are different. The SEL rate significantly increases with increasing supply voltage. Also an increase in operating temperature results in an increase in SEL rate, see Fig. 1.13. In contrast, SER decreases with increasing supply voltage and generally does not show a significant temperature dependency. Because the supply voltage has been reduced to levels below 1.3 V, SEL has

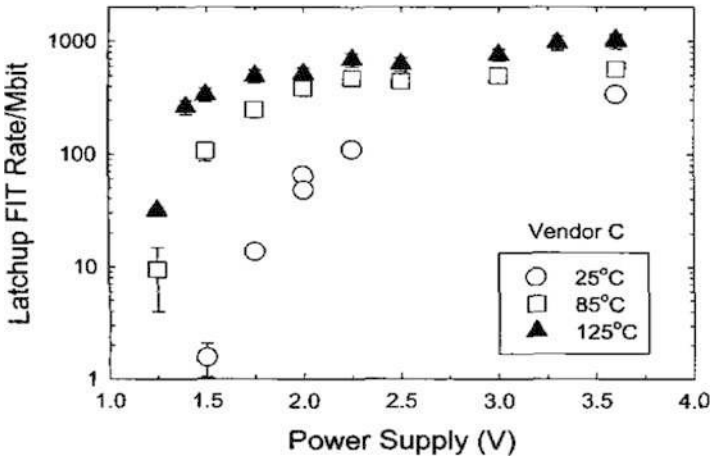


Fig. 1.13 Neutron-induced latchup rate of a 3.3-V SRAM in 0.25-μm (figure from [31])

become a much less severe risk for the most advanced process technologies. For processes with higher supply voltages than about 1.3 V, the SEL risk can be controlled by technology modifications and by applying dedicated design rules.

An advantage of SOI, compared to bulk CMOS, is that it is immune to both electrical and radiation-induced latchup. This is because the parasitic thyristor structure that is vulnerable to latchup does not exist in SOI, since the BOX layer is isolating the transistor from the substrate.

1.4.6 Future Trends

Until about one decade ago, system-level SER was growing mainly due to an increase in the FIT/Mbit of SRAM and DRAM modules in the system, either embedded or stand-alone. This increase was driven by the reduction in feature sizes and especially by the lowering of the supply voltage. With ongoing technology scaling, the FIT/Mbit of SRAMs and sequential elements, the most vulnerable parts in modern process technologies, started to saturate. As a result, the increase in chip-level SER was more and more driven by the increase in the bit count from one product generation to the next. As discussed above, in the most advanced process the FIT/Mbit is actually decreasing. This may result in a saturation of the system-level SER in the future. However, this will depend on the scaling of the SER of sequential and combinational logic. If these contributions do not level off sufficiently, they may dominate the system-level SER in future product generations. However, recent findings do not seem to support this assumption [30]. Compared with memories, which have regular structures that are suitable for protection, reducing SER in logic circuitry is more complicated and generally comes with relatively large design overheads.

Another issue is the ratio between the alpha-particle- and neutron-induced contributions to the SER. In the late 1970s and early 1980s, most of the SER-related industrial problems were caused by alpha particles, often due to a contamination source that had not been anticipated. Since then, careful material engineering has generally resulted in a better control of the contamination risk. Also, the typical concentration of radioactive impurities has been significantly reduced for the products for which SER is a potential reliability risk. Because of material improvements, the alpha-particle emission rates went down from levels as high as 100 α/h cm^2 to levels below 0.001 α/h cm^2 [26]. However, with decreasing critical charges, the alpha-particle-induced contribution may return as a dominant source of SER. This is illustrated in Fig. 1.14: the contribution from alpha particles to the SER is relatively large if the critical charge is small. This is because alpha particles on average generate much less charges in silicon than neutrons. Therefore, when technology scaling results in a lower Q_{crit}, the alpha-SER contribution becomes more important. As a result, alpha particles could increase the contributions of combinational logic, which used to be vulnerable to neutrons only because critical charges are scaling down into the alpha regime.

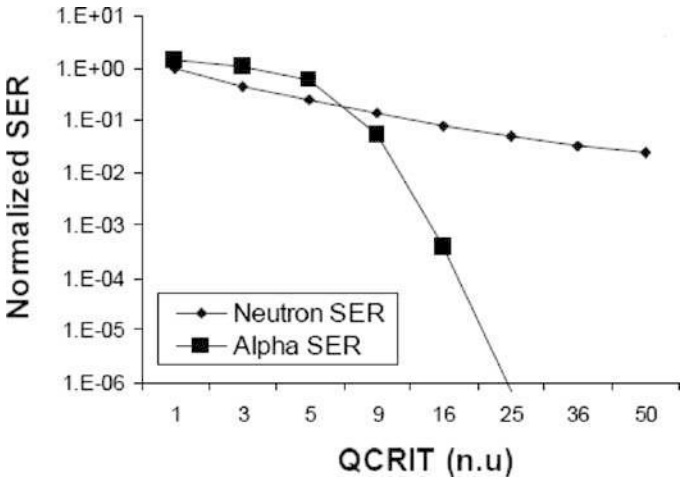


Fig. 1.14 Alpha-particle- and neutron-induced SER contributions as a function of the critical charge (figure from [17])

During the last years, the focus of research on soft errors is more and more shifting from the cell or module level to the chip or system level. The FIT/Mbit of the design elements is important, but it is not the only factor determining the failure rate that is observed at the system level. In order to perform a proper risk assessment, SER should be investigated at all abstraction levels of a system design, including both hardware and software. The same is true for mitigating the SER risk with design changes, if analysis shows that this is necessary. An overview of hardware- and software-based mitigation techniques can be found in Chaps. 8 and 9 of this book, respectively. Whether the SER of a system is an important reliability risk or not strongly depends on its application. For a mobile phone or a car radio, different SER levels are acceptable than for a server or a safety-critical application. Therefore, the application mission profile of the system plays an essential role in the risk assessment. The refresh rate of the applied memories and registers is an important factor. To guarantee the reliable operations of modern products, a built-in reliability (BiR) approach is necessary. Chapter 10 of this book discusses an approach to build a reliable electronic system and to verify its SER performance.

In the last 30 years, many investigations have been spent to study the soft-error subject and many successes have been booked. This trend will continue in the coming years, as SER is becoming a potential reliability risk for more and more applications at sea level. Continuous research is needed on solutions to reduce SER. Such studies have to be performed at different abstraction levels, in order to be able to select the solution with the highest return on investment for a specific product. In the future, it will be increasingly important to have solutions that not only reduce SER, but also address other risks, such as process variability and transistor aging. For example, wear-out mechanisms may cause an increase in the threshold voltages

of the transistors and a reduction of their drive currents. As a result, critical charges decrease and circuits become more susceptible to soft errors.

1.5 Conclusions

Radiation effects, until then an issue affecting space applications only, were catapulted into the world of reliability physics of electronic systems by a milestone paper by May and Woods in the late 1970s. Since then, much empirical evidence for these failure mechanisms has been provided, not seldom related to problems observed in commercial products. Nowadays, future trends seem less dramatic than what was predicted in the past. The soft-error rates per bit of the most sensitive components (SRAM, DRAM, sequential logic) have saturated or even decrease in the most advanced process technologies. However, because of the increasing system sizes and complexities, ongoing investigations will be necessary to reduce the reliability risk for electronic systems.

References

1. JEDEC Standard JESD89A, “Measurement and reporting of alpha particle and terrestrial cosmic ray-induced soft errors in semiconductor devices”, Oct. 2006.
2. D. Binder, E.C. Smith, and A.B. Holman, “Satellite anomalies from galactic cosmic rays”, IEEE Trans. Nucl. Sci., vol. NS-22, no. 6, pp. 2675–2680, 1975.
3. T.C. May and M.H. Woods, “A new physical mechanism for soft errors in dynamic memories”, in Proc. Int’l Rel. Phys. Symp. (IRPS), pp. 33–40, 1978.
4. J.F. Ziegler et al., “IBM experiments in soft fails in computer electronics (1978–1994)”, IBM J. Res. Dev., vol. 40, no. 1, pp. 3–18, 1996.
5. J.F. Ziegler and W.A. Lanford, “Effect of cosmic rays on computer memories”, Science, vol. 206, pp. 776–788, 1979.
6. J.F. Ziegler and W.A. Lanford, “The effect of sea level cosmic rays on electronic devices”, J. Appl. Phys., vol. 52, no. 6, pp. 4305–4311, 1981.
7. T.J. O’Gorman, “The effect of cosmic rays on the soft error rate of a DRAM at ground level”, IEEE Trans. Electron Devices, vol. 41, no. 4, pp. 553–557, 1994.
8. T.J. O’Gorman, J.M. Ross, A.H. Taber, J.F. Ziegler, H.P. Muhlfeld, C.J. Montrose, H.W. Curtis, and J.L. Walsh, “Field testing for cosmic ray soft errors in semiconductor memories”, IBM J. Res. Dev., vol. 40, no. 1, pp. 41–49, 1996.
9. C. Lage, D. Burnett, T. McNelly, K. Baker, A. Bormann, D. Dreier, and V. Soorholtz, “Soft error rate and stored charge requirements in advanced high-density SRAMs”, in Proc. IEEE Int. Dev. Meet. (IEDM), pp. 821–824, 1993.
10. R.C. Baumann, R.C. Baumann, T.Z. Hossain, S. Murata, and H. Kitagawa, “Boron compounds as a dominant source of alpha particles in semiconductor devices”, in Proc. Int’l Rel. Phys. Symp. (IRPS), pp. 297–302, 1995.
11. R.C. Baumann, “Soft errors in advanced semiconductor devices – Part I: The three radiation sources”, IEEE Trans. Device Mater. Reliab., vol. 1, no. 1, pp. 17–22, 2001.
12. R.C. Baumann and E.B. Smith, “Neutron-induced boron fission as a major source of soft errors in deep submicron SRAM devices”, in Proc. Int’l Rel. Phys. Symp. (IRPS), pp. 152–157, 2000.

13. Forbes Magazine: Daniel Lyons, "Sun Screen", Nov. 13, 2000; http://members.forbes.com/global/2000/1113/0323026a.html.
14. Cisco 12000 Single Event Upset Failures Overview and Work Around Summary, April 15, 2003; http://www.cisco.com/en/US/ts/fn/200/fn25994.html.
15. N. Cohen, T.S. Sriram, N. Leland, D. Moyer, S. Butler, and R. Flatley, "Soft error considerations for deep-submicron CMOS circuit applications", in Int'l Electron Devices Meeting (IEDM) Tech. Dig., pp. 315–318, 1999.
16. R. Baumann, "The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction", in Int'l Electron Devices Meeting (IEDM) Tech. Dig., pp. 329–332, 2002.
17. N. Seifert, P. Slankard, M. Kirsch, B. Narasimham, V. Zia, C. Brookreson, A. Vo, S. Mitra, B. Gill, and J. Maiz, "Radiation-induced soft error rates of advanced CMOS bulk devices", in Proc. Int'l Rel. Phys. Symp. (IRPS), pp. 217–225, 2006.
18. P. Roche and G. Gasiot, "Impacts of front-end and middle-end process modifications on terrestrial soft error rate", IEEE Trans. Device Mater. Reliab., vol. 5, no. 3, pp. 382–396, 2005.
19. E.H. Cannon, M.S. Gordon, D.F. Heidel, A.J. KleinOsowski, P. Oldiges, K.P. Rodbell, and H. H.K. Tang, "Multi-bit upsets in 65 nm SOI SRAMs", in Proc. Int'l Rel. Phys. Symp. (IRPS), pp. 195–201, 2008.
20. T. Heijmen and B. Kruseman, "Alpha-particle induced SER of embedded SRAMs affected by variations in process parameters and by the use of process options", Solid State Electron., vol. 49, no. 11, pp. 1783–1790, 2005.
21. T. Heijmen, "Spread in alpha-particle-induced soft-error rate of 90-nm embedded SRAMs", in Proc. Int'l On-Line Testing Symp. (IOLTS), pp. 131–136, 2007.
22. K. Osada, K. Yamaguchi, Y. Saitoh, and T. Kawahara, "Cosmic-ray multi-error immunity for SRAM, based on analysis of the parasitic bipolar effect", in Proc. VLSI Circuit Digest of Technical papers, pp. 255–258, 2003.
23. J.F. Ziegler, M.E. Nelson, J.D. Shell, R.J. Peterson, C.J. Gelderloos, H.P. Muhlfeld, and C.J. Montrose, "Cosmic ray soft error rates of 16-Mb DRAM memory chips", IEEE J. Solid-State Circuits, vol. 33, no. 2, pp. 246–252, 1998.
24. L. Borocki, G. Schindlbeck, and C. Slayman, "Comparison of accelerated DRAM soft error rates measured at component and system level", in Proc. Int'l Rel. Phys. Symp. (IRPS), pp. 482–487, 2008.
25. T. Heijmen, P. Roche, G. Gasiot, K.R. Forbes, and D. Giot, "A comprehensive study on the soft-error rate of flip-flops from 90-nm production libraries", IEEE Trans. Device Mater. Reliab., vol. 7, no. 1, pp. 84–96, 2007.
26. R.C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies", IEEE Trans. Device Mater. Reliab., vol. 5, no. 3, pp. 305–316, 2005.
27. T. Heijmen and P. Ngan, "Radiation-induced soft errors in 90-nm and below", presentation at 12th Annual Automotive Electronics Reliability Workshop, May 2007.
28. N. Seifert, B. Gill, V. Zia, M. Zhang, and V. Ambrose, "On the scalability of redundancy based SER mitigation schemes", in Proc. Int'l Conf. IC Design Techn. (ICICDT), pp. 1–9, 2007.
29. S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K.S. Kim, "Robust system design with built-in soft-error resilience", IEEE Comput., vol. 38, no. 2, pp. 43–52, 2005.
30. B. Gill, N. Seifert, and V. Zia, "Comparison of alpha-particle and neutron-induced combinational and sequential logic error rates at the 32 nm technology node", in Proc. Int'l Rel. Phys. Symp. (IRPS), pp. 199–205, 2009.
31. P.E. Dodd, M.R. Shaneyfelt, J.R. Schwank, and G.L. Hash, "Neutron-induced latchup in SRAMs at ground level", in Proc. Int'l Rel. Phys. Symp. (IRPS), pp. 51–55, 2003.

Chapter 2
Single Event Effects: Mechanisms and Classification

Rémi Gaillard

Single Event Effects (SEEs) induced by heavy ions, protons, and neutrons become an increasing limitation of the reliability of electronic components, circuits, and systems, and have stimulated abundant past and undergoing work for improving our understanding and developing mitigation techniques. Therefore, compiling the knowledge cumulated in an abundant literature, and reporting the open issues and ongoing efforts, is a challenging task. Such a tentative should start by discussing the fundamental aspects of SEEs before reviewing the different steps that are necessary for creating comprehensive prediction models and developing efficient mitigation techniques.

SEE occurs when the charge resulting from the carriers (holes and electrons) liberated by an ionizing particle (heavy ion, alpha particle, recoils of nuclear interaction between a neutron and an atom of the die, ...), and collected at a junction or contact, is greater than the charge carrying an elementary information in the component.

One way to understand the effects is to use a top-down approach to describe the successive steps involved in the perturbation mechanism:

- Interaction of radiation with matter
- Ionization mechanisms and track structure
- Collection of charges
- Description and calculation of effects on semiconductor structure (junction, device)
- Description and calculation of effects on semiconductor circuits and systems

The main difficulty is that each step is situated in a different physical/description domain and involves different people with different skills (nuclear and solid state physicists, process engineers, circuit designers, system engineers...). So bridges

R. Gaillard (✉)
Consultant, Saint-Arnoult on Yvelines France
e-mail: remi.gaillard@iroctech.com

M. Nicolaidis (ed.), *Soft Errors in Modern Electronic Systems*,
Frontiers in Electronic Testing 41, DOI 10.1007/978-1-4419-6993-4_2,
© Springer Science+Business Media, LLC 2011

between these domains must be built to allow an efficient collaboration between the experts mastering them.

This chapter presents a review of the main aspects of SEEs on digital electronics. Analog and power electronics are not considered in this chapter.

2.1 Introduction

Single Event Effects (SEEs) are induced by the interaction of an ionizing particle with electronic components. Ionizing particles can be primary (such as heavy ions in space environment or alpha particles produced by radioactive isotopes contained in the die or its packaging), or secondary (recoils) created by the nuclear interaction of a particle, like a neutron or a proton with silicon, oxygen or any other atom of the die. SEEs become possible when the collected fraction of the charge liberated by the ionizing particle is larger than the electric charge stored on a sensitive node. A sensitive node is defined as a node in a circuit whose electrical potential can be modified by internal injection or collection of electrical charges. Usually, an information bit is associated with a voltage level.

Today, these effects are considered as an important challenge, which is strongly limiting the reliability and availability of electronic systems, and have motivated abundant research and development efforts in both the industry and academia.

These studies involve a broad domain of physics such as natural radiation environment modeling, radiation interaction mechanisms, ion energy depositions in insulators and semiconductors, charge transport and collection in elementary semiconductor devices such as PN junctions or elementary MOS transistors. The studies involve also electronic engineers that analyze perturbation mechanisms and effects in various cells (gates, flip-flops, registers, and memory cells), perturbation effects in complex integrated circuits with the possibility of multiple errors and single event functional interrupt (SEFI).

Designers must also consider analog components either CMOS or Bipolar that can also be upset (operational amplifiers, regulators, comparators, oscillators).

Finally, system level engineers must investigate the global effects of single events on electronic systems and develop mitigation techniques and test methods to establish and verify the robustness of these systems.

The aim of this chapter is to describe the physical mechanisms that induce SEEs and to define and classify the different ways they alter circuit operation. A top-down presentation will be given from environment to effects in complex function.

First, the natural radiation environment at ground level and in the atmosphere will be quickly recalled. A recall of nuclear physics, interaction mechanisms, and energy deposition, is given.

Then, solid-state physics and semiconductor physics are considered in order to develop transient photocurrent models related to the collection of carriers in junctions.

Finally, the effects observed in electronic devices will be defined and classified.

2.2 Background on Environment, Interaction Mechanisms, and Recoil Energy Loss

2.2.1 Natural Radiation Environment

At high altitudes in the atmosphere and at ground level, nuclear radiation can be found within a broad domain of energies. This radiation includes primary radiation from galactic or extragalactic origin with a low flux and secondary radiation resulting from the interaction of the primary radiation with the atmosphere content (nitrogen, oxygen, carbon dioxide, rare gas).

When galactic cosmic ray (GCR) particles reach the earth's atmosphere, they collide with the nuclei of nitrogen and oxygen atoms and create cascades of secondary radiation of different kinds, including leptons, photons, and hadrons.

At the time of writing this book, the neutrons are considered as the most important source of radiation as far as SEEs are considered. Nevertheless, we must also remember that protons and pions can also induce SEEs, and direct ionization of protons may be a concern in future electronics as circuits are further shrinking and become increasingly sensitive to SEE.

The flux of neutrons varies with altitude (the atmosphere attenuates both the cosmic rays and the neutron flux), latitude (due to the variation of the earth's magnetic field shielding efficiency from equator to the poles), and to a lesser extent with longitude. The flux of neutrons can be modulated by solar activity and solar eruptions.

Recently, analytical models describing these variations have been presented in Jedec JESD89A [1] and discussed in Chap. 3.

Measurements have been performed using different monitors or spectrometers.

The energy spectrum is generally considered to be independent of altitude. It extends from thermal energies up to 10 GeV. In first approximation, the differential energy spectrum $\mathrm{d}n/\mathrm{d}E$ can be considered to be proportional to $1/E$. This means that we get about the same number of neutrons in the energy range 1–10 MeV, 10–100 MeV, and 100 MeV–1 GeV. For $E > 1$ GeV, the slope $\mathrm{d}n/\mathrm{d}E$ can be considered with a good approximation to be proportional to E^{-2}.

More precisely, the neutron spectrum has three major peaks, thermal energy peak, evaporation peak around 1 MeV, and cascade peak around 100 MeV.

More information on the shape of the spectrum can be found in [2, 3]. The recently developed QARM model is described in [4].

The reference flux is taken at New York City at sea level [1] to be 13 n/cm^2 for $E > 10$ MeV. More details are given in Chap. 3.

An other important contribution to soft errors comes from alpha particles (helium nucleus built with two neutrons and two protons) emitted by radioactive elements found at trace levels in different materials close to the sensitive nodes of the electronic devices. These alpha emitters are elements from the three natural decay chains called Thorium, Radium, and Actinium decay chains. Secular

equilibrium[1] is sometimes assumed, but this hypothesis may not apply in practical cases if purification methods are applied to eliminate some elements in the radioactive chain. The energy of alpha particles is in a domain between 3 MeV and 9 MeV. Alpha emitters from the Thorium decay chain are given in the following table.

Elements	232Th	228Th	224Ra	220Rn	216Po	212Bi	212Po
Energy (MeV)	4.08	5.52	5.79	6.40	6.91	6.21	8.95
Half lifetime	1.410^{10} years	6.75 years	3.63 days	55.6 s	0.145 s	60.55 mn	299 ns

Important efforts are involved in reducing the impurities concentration and characterizing the alpha emission rate from materials used in packaging. Alpha sensitivity is calculated for a flux of 10^{-3} alpha/cm^2/h.

2.2.2 *Neutron Interaction with Matter: Production of Energetic Recoils*

Neutrons can interact with the nucleus either by elastic and nonelastic interaction and give a part of its energy as kinetic energy to the recoils. In some materials such as ^{10}B low energy neutron capture will give an exoenergetic reaction, but generally energy is either conserved (elastic collisions) or a fraction of it is changed in mass so that the total kinetic energy of the recoils is lower than the kinetic energy of the incident neutron.

2.2.2.1 Interaction Probability

The interaction probability per nucleus is given by the cross-section expressed usually in barn/nucleus. The cross-section is the effective area of the nucleus for interaction with a neutron. One barn equals 10^{-24} cm^2. In silicon, the total neutron cross-section decreases from 1.95 barn at $E = 40$ MeV–0.6 barn at $E = 200$ MeV.

To obtain the mean free path without interaction of neutrons in silicon, the number n of nucleus per cm^3 is first calculated. By considering the density d in g/cm^3, the atomic mass 28 g of 1 Avogadro of nucleus $d = 2.33$ g/cm^3, and the Avogadro number $N = 6.02 \times 10^{23}$, we find $n = 6.02 \times 10^{23}/(28/2.33) = 5 \times 10^{22}$ A/cm^3.

For an interaction cross-section value of one barn the mean free-range is:

$$1/\left(10^{-24} \times 510^{22}\right) = 20\,\text{cm}.$$

[1]The state where the quantity of a radioactive isotope remains constant because its production rate (due, e.g., to decay of a parent isotope) is equal to its decay rate.

So, as it is well known, neutrons can cross an important thickness of matter without interacting. The cross-section of interaction of one neutron with an energy of about 100 MeV (with $\sigma = 1$ barn), in a small cube with a side of 1 μm is then:

$$V \times n \times \sigma = 10^{-12} \times 510^{22} \times 10^{-24} = 510^{-14}\,\mathrm{cm}^2.$$

This value gives the order of magnitude of the upset cross-section per bit that will be obtained for SEU in SRAMs.

Because silicon is the main constituent of electronic devices, we can consider that silicon is the main target material. But for more accurate estimations, we need to take into account all the materials that may exist in the structure of an electronic device. As it is discussed later, we need to consider all the recoils that may come close to or pass through a sensitive node. These recoils must be produced at a distance from the sensitive node that is lower than their range.

2.2.2.2 Elastic Interaction

In the elastic process, the recoil nucleus is identical to the target nucleus. In the collision, the total kinetic energy and the momentum of the neutron-target nucleus system are conserved. A fraction of the energy of the neutron is given to the nucleus. The maximum energy that can be given to the recoil is:

$$E_{\max} = 4 \times E \times \frac{A}{(A+1)^2},$$

where A is the atomic number of the target nucleus. The probability to give a given energy between 0 and $E_{\max}$ is obtained by the differential elastic cross-section.

Most of the silicon recoils produced by elastic collisions have a low energy. For example, with an initial neutron energy $E_n = 125$ MeV, only 5% of the interactions give a recoil with energy larger than 0.27 MeV and 0.1% give a recoil energy larger than 1.5 MeV.

2.2.2.3 Nonelastic Interaction

Nonelastic interactions group all the interactions that result in a fragmentation of the nucleus in two or more recoil fragments. Generally, the lighter recoil is indicated to describe the reaction: (n,p), (n,α), (n,d). The heavier element is obtained by the equilibrium of the number of neutrons and protons before and after the reaction. With ^{28}Si as the target nucleus, (n,p) reaction results in a proton and Al recoil, (n,α) reaction results in He and Mg recoils.

The incident energy of the neutron diminished by the mass variation is shared between the secondary particles and the main recoil nucleus. These reactions require that the incident neutron energy is larger than the threshold energy of

each reaction. For a given neutron energy, different reactions are possible but their relative probability varies with the neutron energy.

2.2.2.4 Inelastic Collision: (n,n′)

In this reaction, the incident neutron is absorbed in the target nucleus and a short time later a neutron is ejected with a lower energy, sharing a part of the total kinetic energy with the recoil target nucleus. The emission is considered to be isotropic.

Most of recoils with $E_r > 2$ MeV come from nonelastic collision as shown by Chadwick [5, 6].

2.2.2.5 Pion Creation

Above about 250 MeV, neutrons can interact directly with nucleons in the silicon nucleus to form pions. Charged pions lose energy through ionization and by reacting with additional silicon nuclei or decaying in about 10^{-28} s to form charged secondary particles that also deposit charge in the silicon.

When more than two fragments are created, the repartition of energies and angles can be difficult to evaluate (multibody reactions). More details are given in Chap. 4.

2.2.2.6 Nuclear Data Base

The atmospheric neutron energy spectrum covers a broad energy range from meV (10^{-3} eV) to 10 GeV. A detailed knowledge of the recoil products is needed to calculate the range and density of ionization. So the energy spectrum is divided in different energy bins. For each neutron energy, the distribution of the nature and the energy of the different recoils are calculated.

In this way, a database is built to allow further use by Monte–Carlo Sampling in this distribution.

To build the database, a Monte–Carlo interaction code is needed. These codes must calculate the energy and direction of all the recoils. Wrobel [7] has developed the code MC-RED to develop such a data base. Chadwick [5, 6] has performed the extension of MCNP code previously limited to 20–150 MeV.

Data bases must be built for all the materials that can be met in the vicinity of the sensitive node. In each SEEs prediction codes, the data base is one of the most important parts that define the precision of the prediction. Usually, Si and O target nucleus are considered, but W used in vias and Cu used in metalization layers are also considered.

Tukamoto [8] has detailed in such a way that database can be obtained in the full spectrum energy domain. He used the JENDL-3.3 library to obtain the double-differential cross-sections (DDXs) for light charged particles (p and α) and

all recoils(^{24,25}Mg, ^{27,28}Al, and ^{27,28}Si) in the n+ ^{28}Si reaction at energies between 2 and 20 MeV. The data for energies between 20 and 150 MeV were taken from the LA150 library [5, 6] in which the DDXs of all recoils are included. Tukamoto has calculated the cross-sections for energies above 150 MeV using the QMD code [9] plus statistical decay model (GEM [10]) calculation.

The total deposited energy, in terms of kinetic energy transferred to recoils, is also an important parameter. This term was considered initially in the Burst generation rate (BGR) method to predict the Soft Error Rate (SER) [11]. It represents a worst case approach for SER calculation because all the energy transferred to recoils is assumed to be transformed in ionization and totally collected at the sensitive node.

The energies of the recoils are for a large part lower than 5 MeV.

If for 100 MeV neutrons, ^{28}Si recoil due to nonelastic collision is considered, it can be shown that 90% of the silicon recoils have their energy lower than 2.5 MeV and 99% their energy lower than 5 MeV. But light particles such as He nucleus (Alpha particles) have a higher energy. For 100 MeV neutrons, 10% of recoils have their energy greater than 16 MeV, and 1% have their energy greater than 31 MeV.

For $E_n = 20$ MeV, the emitted recoils are separated in light recoils (protons (n,p), deutons (n,d), tritons(n,t), ^{3}He, ^{4}He (n,α)), and heavy recoils (^{21}Ne, ^{27}Al, ^{28}Al, ^{24}Mg, ^{25}Mg, ^{26}Mg, ^{28}Si, ^{24}Na). No recoils with $5 < A < 21$ are produced. For higher energies, most of the isotopes with A values from 1 to 28 are present.

The creation of a nuclear data base is one of the most important steps in establishing a prediction tool to calculate the sensitivity of devices.

The nuclear data base gives also the angle of emissions when multiple particles are generated in a single interaction. The energies and emission directions are correlated to the incident neutron energy and direction.

A complete nuclear data base would give information on recoils resulting from all the materials that are at a distance from the sensitive zone lower than the maximum range of the recoils.

In particular recoils from oxygen (in SiO_2), W used in plugs, Cu in metalization, and also high K dielectrics used in the gate of MOS transistors, must be considered. Their relative contribution to the SER of a device is related to the technology used (number and thickness of metalization levels, structure of the cells, ...).

One important problem is the way to use the important volume of data generated in the data base and to calculate the sensitivity of the devices.

2.2.3 *Recoils: Ionization and Ranges*

The different recoils produced in an interaction extend from proton or neutron to the nucleus of the target atom. Neutrons produced in the reaction are not considered since their probability of interaction in the vicinity of their creation is very small.

The ions lose their energy by coulomb interaction with the electrons of the target atoms and at the end of their range by elastic collisions with the atoms [48].

The energy loss as a function of energy is given by codes such as SRIM [12] or MSTAR [13].

The most important parameters are the total or projected range of the ion, the energy loss by unit length, the velocity of the ion and finally the transverse profile of the ionization density in the ion track.

The ion range will determine the volume around a sensitive node where a neutron interaction may create a recoil able to reach this node.

The energy loss per unit length gives the density of the electron-hole pairs that can be collected in the space-charge layer and diffusion length of a PN junction

The transverse profile of the track is to be compared to the junction width or the channel length of a device.

2.2.3.1 Ranges of Recoils

We must separate the light ions (alpha, protons) which have generally a higher energy and a very long range from the heaviest ions (Si, Al, Mg). Range of silicon recoils, in a silicon lattice, with an energy <10 MeV, calculated with SRIM2008 [12] is given below. For $E < 5$ MeV, the range is less than 3 µm.

This range is short but much larger than the dimensions of transistor dimensions in recent technology nodes so that even heavy recoils can reach several sensitive nodes (Fig. 2.1).

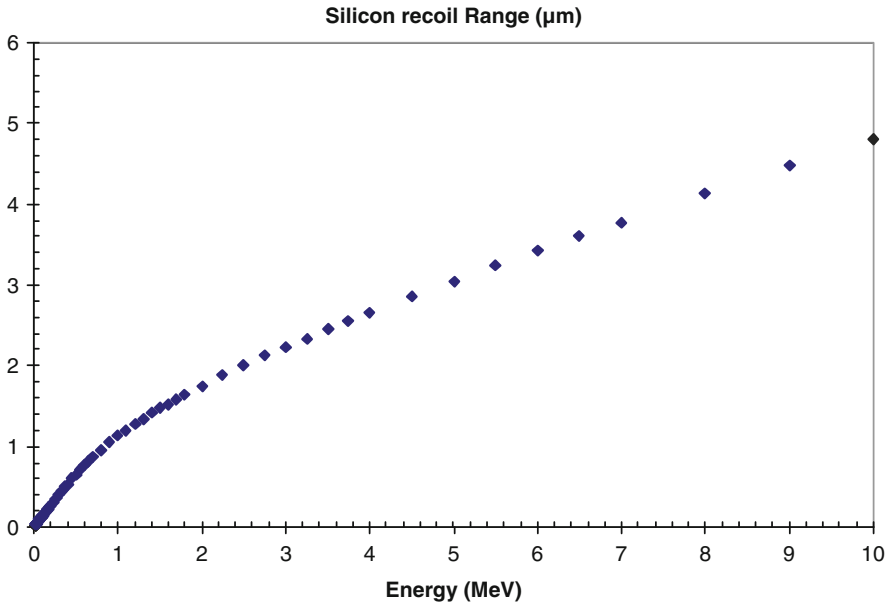


Fig. 2.1 Range of silicon recoil in a silicon lattice

2.2.3.2 Velocity of Ions

In the energy range of recoils produced by atmospheric neutrons, classical theory can apply and the velocity is related to energy by $v = \mathrm{SQRT}(2E/m)$ with $m = A \times \mathrm{AMU}$, A being the atomic number and AMU the Atomic Mass Unit.

1 AMU $= 1.66054 \times 10^{-27}$ Kg.

For 1 MeV Si ion, $v = 1.3$ μm/ps, and for 1 MeV alpha particle $v = 3.47$ μm/ps.

2.2.4 Ionization

Ionization is due to coulomb interaction with the electrons of silicon atoms of the recoil ions with effective charge Zeff. High energy electrons (δ rays) are created which lose their energy by further ionization or phonon excitation in times of 1–100 fs.

Finally, a mean of 3.6 eV (1 eV $= 10^{-19}$J) is needed to create a free hole-electron pair in silicon. The energy transfer (LET) is the energy deposited per unit length given in MeV $\times$ cm^2/mg, and describes the average energy loss by the ion per unit length of the transverse material. The LET can be converted in charge per unit length (pC/μm or fC/μm) to use units more convenient to the electronic designer because they can be compared to physical dimensions of the device and to the charge stored at critical nodes.

To perform this conversion, first the energy loss per unit length is calculated. In silicon (silicon density equals 2.32 g/cm^3), we have:

$$1\,\mathrm{MeV}/\mu\mathrm{m} = 4.31\,\mathrm{MeV}/\mathrm{mg}/\mathrm{cm}^2.$$

Then, substituting 1 MeV/mg/cm^2 by 10 fC/μm we obtain:

$$1\,\mathrm{MeV}/1.610^{-19} \times 10^6/3.6\mathrm{C}/\mu\mathrm{m} = 44\,\mathrm{fC}/\mu\mathrm{m}.$$

For each recoil, the LET varies with its energy. First, the LET increases with recoil energy until reaching a maximum called the *Bragg peak* and then decreases with increasing energy. For the different ions that may result from the spallation of silicon atoms, the highest LET is obtained for silicon which is the heaviest possible recoil in a silicon target. The max LET can reach 15 MeV/mg/cm^2 at the Bragg peak. This value is often used as a rough criteria to separate devices sensitive to neutrons (threshold LET < 15) from devices that are not sensitive (threshold LET > 15), by using heavy ions test results. But this approach can be in error if heavier recoils (from Cu metalization or W plugs) can reach the sensitive nodes (Fig. 2.2).

LET is an important parameter to quantify the sensitivity of devices.

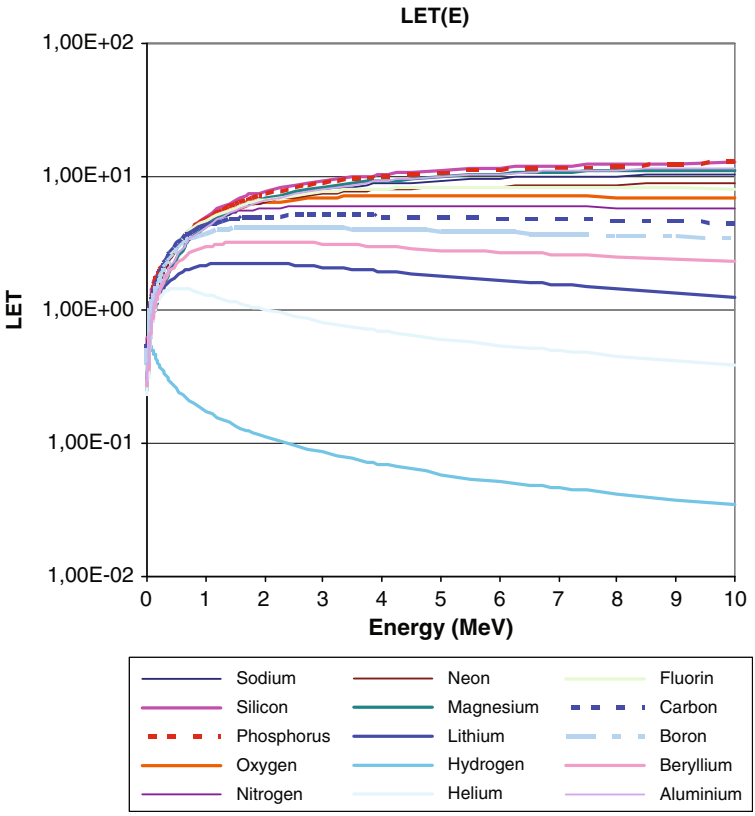


Fig. 2.2 LET(E) in MeV × cm^2/mg of recoils ($Z = 1, 15$) in silicon for $E < 10$ MeV

The range is related to the initial energy E and the LET by the relation:

$$R = \int_{E_{\max}}^{0} \frac{\mathrm{d}x}{\mathrm{d}E}\mathrm{d}E$$

When a recoil is created at a distance d from a sensitive node, the energy of the particle when it reaches the vicinity of the node is given by:

$$E_d = E_0 - \int_0^d \frac{\mathrm{d}E}{\mathrm{d}x}\mathrm{d}x$$

If the initial energy is greater than the Bragg peak energy, the LET increases along the track until the Bragg peak energy is reached and then decreases.

Table 2.1 Energy and LET at the Bragg peak for recoils with $Z = 1$–8

Element	H	He	Li	Be	B	C	N	O
Energy (MeV)	0.05	0.45	1.65	2	2.1	2.8	3.9	5
LET (MeV × cm^2/mg)	0.538	1.45	2.26	3.24	4.19	5.13	6.04	7.17
LET (fC/μm)	5.49	14.8	23.1	33.1	42.8	52.4	61.6	73.2

This effect can be important for protection layers against alpha particles. If the layer is not thick enough to stop the particle, the LET may be greater near a sensitive node than without protective layer.

The energy of different recoils at the Bragg peak in silicon and the associated LET are given in Table 2.1.

For protons (H nucleus), at the Bragg peak (situated at very low energy near 50 keV), the LET reaches 5.5 fC/μm. Then, it decreases rapidly to 2.6 fC/μm at 500 keV and 1.8 fC/μm at 1 MeV. So protons may contribute to upset by direct ionization at low energy when the collection length is greater enough (>200 nm) and when the critical charge is less than 1 fC. This point will be discussed for SRAM devices.

2.2.4.1 Track Diameter

The initial radial variation of electron-hole pair density along the ion track is an important input in the semiconductor-simulation codes to calculate the collected charge at the sensitive nodes. Generally, a Gaussian shape is assumed with an initial diameter that will broaden with time. In the past, when devices dimensions were in the micron range, the precise knowledge of the track diameter at $1/e^2$ value was not necessary. But with 60 nm or 45 nm design rules a precise knowledge of the radial variation becomes mandatory.

In a recent work, Murat et al. [14] use Monte–Carlo techniques to study ion transport and secondary electron transport up to a remaining kinetic energy of 1.5 eV (very close to the energy gap of 1.12 eV) but their work concerns high energies >10 MeV/AMU, that is >280 MeV for silicon recoils and > 40MeV for alpha particles. Kobayashi [15] performed similar calculations with the code GEANT4 but with a higher value for electron energy cut-off limit. Zhu [16] has found that the maximum range of secondary electrons (δ rays) is about 20 nm for 1.5 MeV alpha particles. More work is needed in this domain to accurately describe the ionization track from its creation to the first picoseconds.

2.2.4.2 Generation Time Profile

The generation of free carriers follows the ion penetration (at a velocity related to its energy and its mass), but sometimes it is considered as instantaneous with the implicit assumption that the response of the device is much slower than the carrier generation time. As shown by Zhu [16], this assumption must be verified and

eventually modified for low energy ions for recoils generated by ^{10}B fission with thermal neutrons.

2.2.5 Conclusion

Radiation is present in the natural environment. The flux of particles varies with altitude, latitude, and to a lesser extent with longitude. The particles have a broad energy spectrum and present several interaction mechanisms that result in energy deposition in the materials. Some impurities present in semiconductors or package materials are radioactive and emit alpha particles that can reach sensitive zones.

In electronic devices, single particles may deposit enough ionizing energy to upset components. To understand SEEs, the particle interaction mechanisms must be quantified and the recoils that are produced by the nuclear spallation reactions must be well identified by their nature, energy, and direction. The loss of energy of these recoils per unit length and the ionization density along the radius of the track are important input parameters for accurate modeling of SEEs.

2.3 Single Event Effects in Electronic Devices and Systems

2.3.1 Single Event Effects Definition

SEEs in Electronic Devices group all the possible effects induced by the interaction of one nuclear particle with electronic components.

These effects are classified in hard errors and soft errors as already presented in Chap. 1.

Hard errors are non-recoverable errors. Soft errors may be recovered by a reset, a power cycle or simply a rewrite of the information.

Hard errors concern Single Event Burnout (SEB) in power MOS, IGBT or power bipolar transistors, dielectric breakdown in gates (SEGR) or in capacitors, microdose-induced threshold voltage variation in MOS transistors.

These hard errors are not discussed in this chapter, but they represent also an important issue in SEEs. These effects were recently reviewed by Sexton [17].

Soft errors include a great variety of manifestations depending upon the device considered.

In analog devices, Single Event Transients (SETs) also called Analog Single Event Transients (ASETs) are mainly transient pulses in operational amplifiers, comparators or reference voltage circuits.

In combinatorial logic, SETs are transient pulses generated in a gate that may propagate in a combinatorial circuit path and eventually be latched in a storage cell (e.g., a latch or a flip-flop).

In memory devices, latches, and registers, single events are mainly called Single Event Upsets (SEU). This corresponds to a flip of the cell state. When for one

particle interaction many storage cells are upset, a Multi-Cell Upset (MCU) is obtained. If more than one bit in a word is upset by a single event, a (Multi-Bit Upset (MBU) is obtained.

For complex integrated circuits, loss of functionality due to perturbation of control registers or clocks is called SEFI. These SEFIs may give burst of errors or long duration loss of functionality. Functionality may be recovered either by a power cycle, a reset or a reload of a configuration register.

In bulk CMOS technology, PNPN parasitic structures may be triggered giving a Single Event Latch-up (SEL). SEL is associated with a strong increase of power supply current. SEL can be destructive by over heating of the structure and localized metal fusion. A SEL needs a power cycle to be deactivated.

2.3.2 Soft Error Rate

The SER is the rate at which soft errors appear in a device for a given environment.

When the particular environment is known or for a reference environment (NYC latitude at sea level), the SER rate can be given in FIT. In semiconductor memory, the sensitivity is often given in FIT/Mb or in FIT/device. One FIT (Failure In Time) is equal to a failure in 10^9 h.

The FIT value is either predicted by simulation or is the result of an experimental error measurement near a neutron generator representing as accurately as possible the real neutron energy spectrum.

The cross-section can be used to calculate SER. If, with a total fluence of N neutrons/cm^2, n errors are obtained in one device, the cross-section is $\sigma = n/N$. With ϕ the particle flux in the real environment, expressed in n/cm^2/h, the FIT value is: FIT value $= (n/N) \times \phi \times 10^9$.

With a cross-section of 5×10^{-14} cm^2 per bit, and $\phi = 13$ n/cm^2/h, the number of FIT/Mb is $5 \times 10^{-14} \times 10^6 \times 13 \times 10^9 = 650$ FIT.

2.3.3 Critical Charge Criteria

The sensitivity of a device must be described by a parameter that allows to simply evaluate the sort of recoils that can disturb the electronic function.

The simplest parameter to consider is the charge collected at a given node and to compare its value to the minimum charge needed to upset the function. This minimum charge is called the *critical charge*.

The simplest approach is to consider the critical charge as the product of the total capacity Ci at a given node by the power supply voltage Vdd: Qc = Ci × Vdd. This approach is very useful to obtain a rough estimation of the critical charge.

In this crude approximation, the collection of charges is considered as instantaneous and the rest of the circuit has no time to react. This approximation can be wrong in particular when carrier collection occurs by a diffusion process.

Considering an inverter, during the time of collection, the ON transistor delivers some current that partially compensates the collected current. This effect was first taken into account by Palau [18] to develop an enhanced model.

A more precise approach to obtain the critical charge is to use a circuit simulator such as SPICE Code and to describe the collected charge by a double exponential current generator of short duration applied at the sensitive node:

$$I_{\text{exp}} = \frac{Q_{\text{tot}}}{\tau_f - \tau_r} \times \left(\exp\left(-\frac{t}{\tau_f} \right) - \exp\left(-\frac{t}{\tau_r} \right) \right).$$

But this current pulse has peak amplitude independent of the voltage at the sensitive node and thus the free carrier collection and function of the voltage is not accurately described.

A best approach but much more complex and time consuming is to use a device simulator working either in full 3D for the complete circuit to be simulated (e.g., 6T SRAM cell) or mixed mode simulation with only one transistor described with a device simulator and the remaining transistors described with a circuit simulator.

These approaches to obtain the critical charge have been described in detail by Dodd [24, 19] and Yamagushi [20].

The charge is calculated by integration of the current pulse crossing the junction. The time limits for the integration are related to the circuit upset mode and the charge must be integrated to a time value that is either the recovery time (when no upset occurs) or the upset time when the device changes state.

The critical charge variation as a function of the time duration of the current pulse has been considered by Jain [21] for a rectangular waveform (Fig. 2.3).

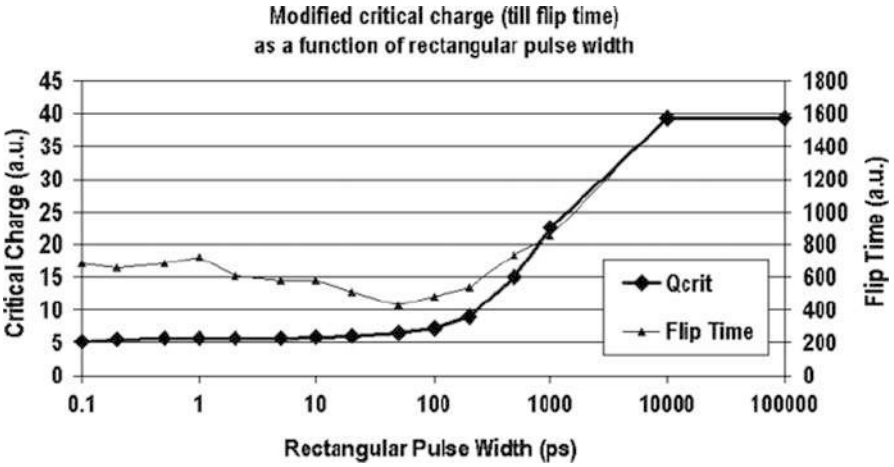


Fig. 2.3 From Jain [21] variation of the critical charge and the flip time as a function of pulse duration. For this particular technology node, the critical charge and flip time increase for $t > 100$ ps

2.3.3.1 Orders of Magnitude of Critical Charge

The total capacity at a given node is related to the technology. In CMOS technology, this capacity at the drain-substrate junction is the sum of gate-drain capacity and drain-substrate capacity. As will be seen later for SRAM devices, the critical charge is reduced to 1–2 fC in most recent technology nodes.

For DRAM and SDRAM, the critical charge remains roughly constant due to sense amplifier limits to 20–30 fC.

2.3.4 Description of Current Pulses for Circuit Simulation

A mathematical equation with a double exponential was given above to describe the waveform of the current pulse in circuit analysis codes. This waveform is arbitrary and mostly adapted to facilitate convergence in the codes. In practice, several waveforms can be obtained depending upon the location and length of the track.

The effects of the free carrier generation along the recoil(s) track(s) can be investigated by studying device response from the simple PN junction to complex devices response with device simulators.

In a PN Junction (such as drain-substrate or drain-well), the current pulse is due to separation of hole-electron pairs by the electric field of the space charge layer (drift component) and by the diffusion of minority carriers toward the space charge layer boundaries where they are accelerated by the electric field.

When the recoil track crosses the depletion layer, some field modification along the track may enhance the collection length by extending the space charge layer in the neutral zone. The so called *funneling effect* is only important for high LET and low doped neutral zones.

When the track does not cross the depletion layer, only diffusion process occurs and gives a slower current pulse. Diffusion is related to the gradient of carriers, and so diffusion of carriers is effective toward all the junctions for which the neutral zone where the track is located is one side of the junction. Analytical expressions of the diffusion component are detailed in Chap. 4.

During diffusion time, a fraction of the generated free carriers recombine before reaching the limits of space charge layers.

To perform a detailed calculation of the current pulse waveform a device simulator is needed. Presently, most of the device simulators use the conventional drift-diffusion equation but more complex treatment will be required in future devices as outlined by Munteanu [22].

The shape of the current pulse varies with the extent of funneling and diffusion present in the charge collection at any given junction.

A double exponential pulse shape is often used to fit the pulses with a characteristic rise time (about 1 ps) and a varying decay time. For example, Bajura [23] presents the comparison of current pulses obtained from 3D device simulation and

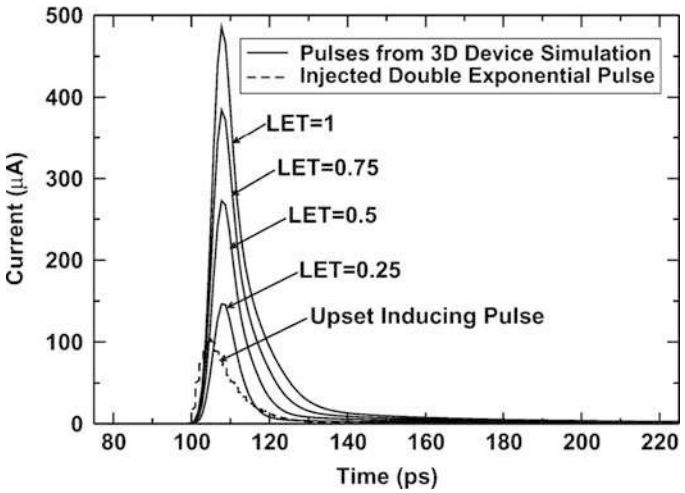


Fig. 2.4 Comparison of 3D simulation pulses with double exponential pulse (From Bajura [23])

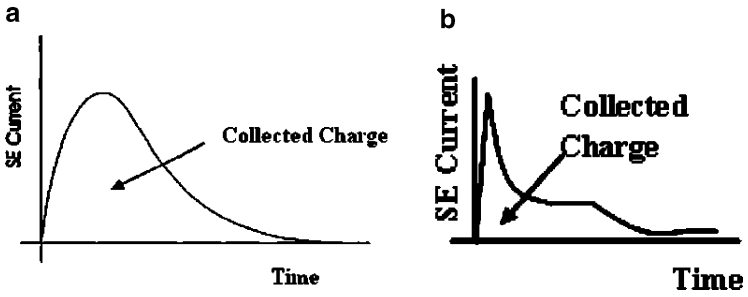


Fig. 2.5 Comparison of double exponential waveform with current waveforms obtained in 3D simulation. (After Dodd [24, 25])

the classical double exponential pulse injected at critical nodes in a 90 nm technology in the Off NMOS transistor (Fig. 2.4).

For older technologies, the charge collection was accurately modeled by a double exponential current waveform (Fig. 2.5a). However, for advanced technologies the current pulse shape becomes much more complex as shown in Fig. 2.5b, as discussed by Dodd [24, 25].

2.4 Sensitivity of Devices

The sensitivity of devices can be investigated by considering the effect of the ionization track on devices with increasing complexity. The simplest device is a reverse-bias PN junction. The current waveforms in PN junctions were discussed

previously. These junctions are often parasitic (drain-substrate, drain-well, well-substrate in bulk CMOS technology) that must be described accurately.

Due to the different locations of impact, track direction, track length, ionization density, and many waveform shapes are expected. So empirical models, taking into account these parameters, must be developed for a given technology, in order to allow the prediction of the behavior of the elementary logic functions such as combinational gates [26].

2.4.1 SET

DSETs (Digital SET) are momentary voltage or current disturbances affecting combinational gates. Although an SET does cause a transient in the gate output struck by the recoil ion, it may propagate through subsequent gates and eventually cause an SEU when it reaches a memory element. Dodd [25] recalls that at least four criteria must be met for a DSET to result in a circuit error.

- The particle strike must generate a transient capable of propagating through the circuit.
- There must be an open logic path through which the DSET can propagate to arrive at a latch or other memory element.
- The DSET must be of sufficient amplitude and duration to change the state of the latch or memory element.
- In synchronous logic, the DSET must arrive at the latch during the latching edge of the clock.

The probability that transient glitches will be captured as valid data in combinational logic increases linearly with frequency because the frequency of clock edges increases. As circuit speeds increase, it is also expected that the ability of a given transient to propagate increases (high speed means faster gates and/or less logic levels per pipeline stage); however, one might also conjecture that the duration of transients decreases. Due to both their greater ability to propagate in high-speed circuits and their higher probability of capture by subsequent storage elements, such as latches, DSETs have been predicted to become a very important issue in deep submicron digital ICs [27, 28].

Mixed-level simulations have been used to study the production and propagation of DSETs in scaled silicon CMOS digital logic circuits [25]. The modeling technique is illustrated in Fig. 2.6. In this figure, the production of SETs in the off-biased n-channel transistor in the struck CMOS inverter is modeled in the 3D device domain, and propagation through an ensuing ten-inverter delay chain is modeled in the circuit domain within the mixed-level simulator. Transients that can propagate without attenuation are obviously a concern, although it must be reminded that propagation of the DSET is only the first of a set of conditions that must be met for a DSET to become an SEU.

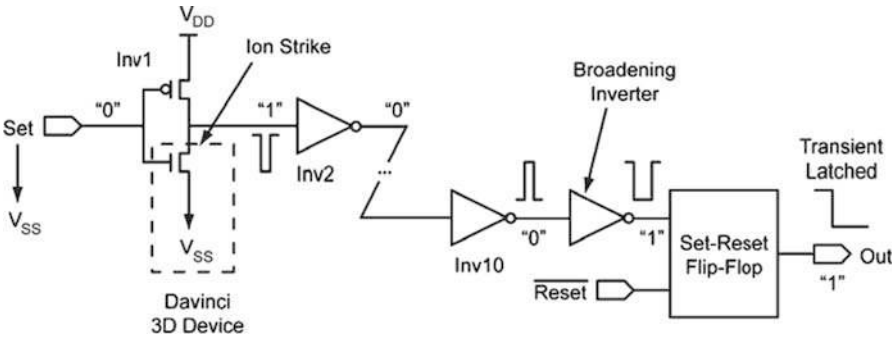


Fig. 2.6 Mixed mode simulation used to study SET propagation in an inverter chain (after Shuming [29])

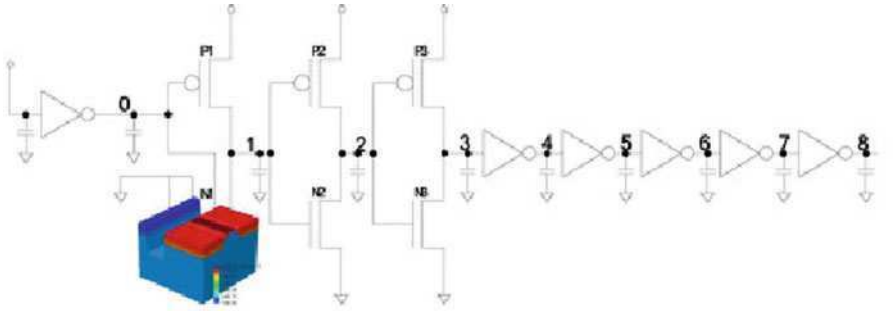


Fig. 2.7 An other example of mixed simulation used to study DSET propagation. Note that an inverter is placed before the inverter described with 3D simulation code (after Dodd [25])

An important advantage of 3D mixed-level simulations compared to circuit simulations is that they can directly predict the ion-induced transients in response to a given particle strike.

At the same time, the simulations calculate the propagation of the transient through the inverter delay chain.

Shuming [29] has investigated the temperature effects on SET in a chain of inverters in 180 nm bulk by using mixed simulation on the structure given below. Dodd [25] uses nearly the same approach. This approach can be used in all combinatorial logics with more complex gates (Fig. 2.7).

2.4.2 SEU

An SEU occurs when an ionizing particle strike modifies the electrical state of a storage cell, such that an error is produced when the cell is read. In an SRAM or

a flip-flop, the state of the memory is reversed. In a DRAM, the charge stored can be slightly modified and interpreted as a wrong value by the read circuitry. In logic circuits, SEUs can also be obtained when a SET propagates through the combinational logic and is captured by a latch or a flip-flop.

SEU is experimentally studied in memories or registers by writing a known content, irradiating the device at a given particle flux and recording the variation of the contents.

Different tests algorithms can be used from the simple write-irradiate-read called also static test to much more complex dynamic test sequences.

2.4.2.1 SRAM

In 6T SRAM cells with two coupled CMOS inverters and two NMOS access transistors, SEUs have been studied by means of full 3D simulations as well as mixed-mode simulations, in order to evaluate the critical charge needed to obtain an upset.

Bajura [23] has studied the minimal-sized SEU-inducing pulses, when applied to the drains of the "off" NMOS and "off" PMOS transistors. He has found a total integrated charge of 1.1 fC and 2.4 fC, respectively. He explains that the difference in minimum injected total charge required for a bit upset to occur is due to the design of the 6T cell. The cross-coupled inverters are highly asymmetric with the pull-down NMOS transistors having much stronger drive strength than the PMOS pull-ups. Under normal operating conditions, this asymmetric design improves the speed at which the SRAM can flip states.

The following figure illustrates the two cross-coupled inverters and the pulse current injected at the drain of the off NMOS in a circuit simulation (Fig. 2.8a, b).

The two inverters output are obtained for the two limiting ionization density just below the upset level and just above the upset level (Fig. 2.9).

On the one hand, critical charge of SRAM is decreasing in successive process nodes due to the shrink in dimensions and the decrease of the power supply voltage. But on the other hand, the area of elementary cells is also decreasing as indicated below (data presented by Bai et al. [30]), resulting in a compensation of the two effects. The decrease of the critical charge implies that an increasing number of interactions produce recoils that can induce SEU. It results that an increasing fraction of recoils becomes effective. For older technologies, only a small part of the reactions were efficient to give SEU, and so a strong increase of the sensitivity was obtained when the critical charge was decreasing for two successive technology nodes. For more recent technologies, the fraction of effective interactions reaches 90% so only a small effect of reduction of critical charge is expected in the future.

The diminution of the cell area will then be the dominant effect and a global reduction of the sensitivity per Mb is expected in the future. The sensitivity trends are more deeply discussed in Chap. 1 (Fig. 2.10).

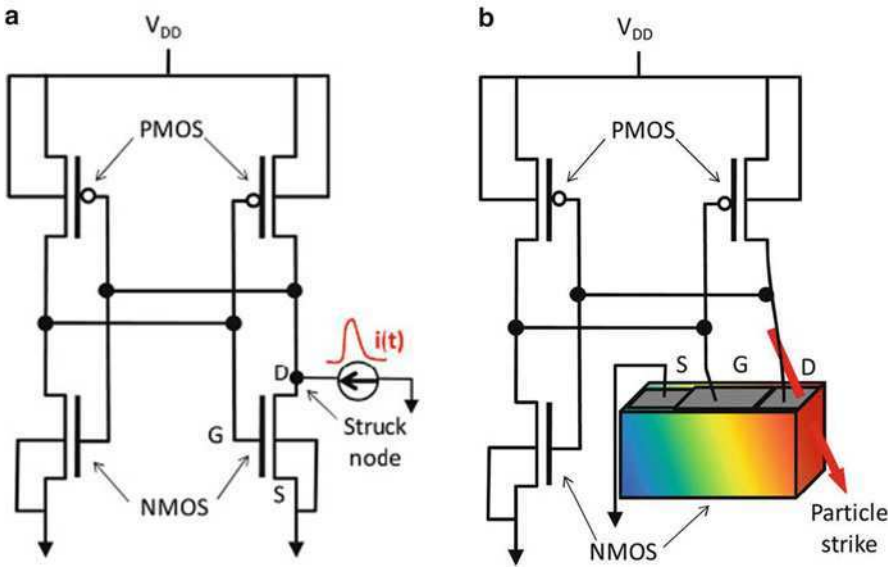


Fig. 2.8 (**a**) Schematic of two cross-coupled inverters in CMOS technology. A current pulse is injected at the drain of the off NMOS transistor. (**b**) Mixed Mode simulation is used to simulate SEU. The Off NMOS is studied by means of device simulation and the remaining transistors are studied with the coupled circuit simulator (after Munteanu [22])

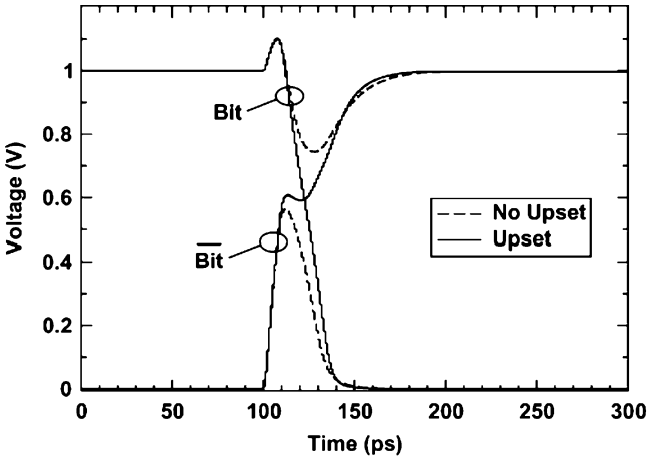


Fig. 2.9 Drain voltage waveforms for ionization density slightly lower (*no upset*) and slightly larger (*upset*) than the ionization upset threshold

2.4.2.2 Upset by Direct Proton Ionization

Low energy protons ($E < 1$MeV) have a LET that varies between 1.8 fC/μm at 1 MeV and 5.5 fC/μm at 50 keV. In SRAM with very low critical charge (<1 fC)

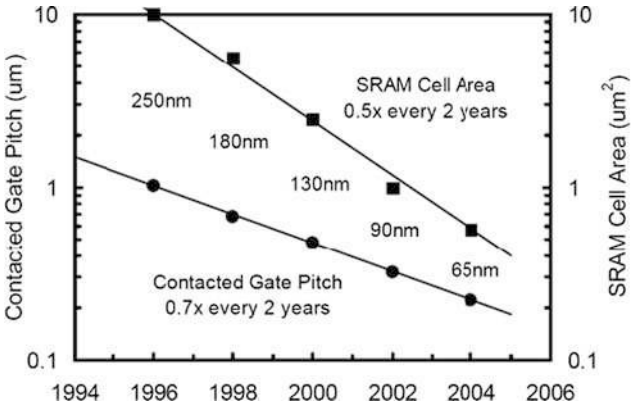


Fig. 2.10 From Bai [30]. SRAM cell areas and contact gate pitch decrease with each technology node. At 65 nm, cell area is about 0.6 μm^2

and for particular proton incident directions where the collection length is greater than 200 nm, upsets are expected. These upsets have recently been observed experimentally [45, 46] in 65 nm SOI SRAMs at high incidence angles.

2.4.2.3 DRAM and SDRAM

A DRAM cell consists of a capacitor and a transistor. The capacitor stores the bit of information, and the transistor isolates the capacitor during nonread or nonwrite times, which is approximately 99% of the time [32]. During this time, the cell experiences subthreshold leakage that causes the DRAM cell to lose its programmed state. Thus, the cell needs to be refreshed, i.e., rewritten, occasionally. During a READ operation, the word-line potential is raised to access the storage node. A sense amplifier compares the potential on the bit line to the potential of a reference bit line on the other side of the sense-amp, and the sense-amp latch sets depending on whether there charge stored in the capacitor exceeds a certain threshold. At the end of the operation, the bit is restored to the cell and the bit lines are charged up to a high level for the next cycle. The bit lines are then allowed to float, making them sensitive to charge collection about 99% of the time.

O'Gorman [33] reported soft errors at ground level in 288 k DRAM and clearly separated the contributions from alpha particles and neutrons.

The critical charge of the cell, defined as the minimum amount of charge that must be collected by the cell to cause a soft error, was in the range of about 70–100 fC.

Scheick [32] indicated that half of the cells have 0 charge on the capacitor when programmed to 1, while the other half have a charge stored when programmed to 1. Now, since a DRAM can only upset by discharge of the capacitor, a DRAM programmed with all ones or zeros will have about half of the cells vulnerable to SEU.

The DRAM vulnerability is related to the technology of the capacitor that stores the information (see Chap. 1).

As clearly indicated by Massengill [34], a degradation of the stored signal to a level outside the noise margin of the supporting circuitry is considered an upset, as it will lead to erroneous interpretation and a resultant error. In a DRAM cell, the current in the source of the NMOS transistor may discharge the capacitor.

In 1987, Hisamoto et al. [35] reported a direct source-drain penetration effect due to alpha-particle strikes in scaled (short channel MOS) DRAM structures. This alpha-particle induced source-drain penetration (ALPEN) effect causes current flow across the normally off access transistor, due to a source-drain shunt effect. While the normal SEE mechanism is to deplete charge from a DRAM cell storage capacitance, the ALPEN mechanism causes the opposite result: the shunting of charge onto the storage node. Thus, a $0 \rightarrow 1$ versus a $1 \rightarrow 0$ mechanism is introduced. In this early study, the ALPEN effect was found to be important for device gate lengths less than 0.3 μm.

2.4.3 MCU and MBU in SRAM and DRAM

SEU is a perturbation of one bit by a single particle. Single Error Correction Codes (ECC) such as Hamming codes that can detect two errors and correct one in a word can be used to mitigate these errors. But if two or more errors are present in a single word, a more elaborated and complex ECC is needed. Multiple errors can be created in devices when a particle crosses the sensitive zones of different cells (drains of blocked MOS transistors) or when the free carriers of the ion track can be collected by different junctions of transistors of several memory cells.

As the elementary cell area is continuously decreasing for successive technology nodes, the probability that recoils have a range long enough to reach different cells increases. So the probability of multiple cell upsets increases. This probability also increases due to an increasing charge sharing probability: neighbor circuit nodes in advanced technologies are closer to each other and can collect charges from a single particle track. The ratio of MCU to SEU and the probability to have a given number or errors in a MCU are important parameters because they allow to determine the efficiency of an ECC and to choose the most appropriate one for meeting the SER requirements of the target application. The geometrical repartition of bits in a word is also an important parameter for determining the fraction of the errors produced by an MCU that are located in the same word (MBU). In particular to avoid MBU, bits in same word are placed as far as possible one from each other by using appropriate scrambling in the memory design.

2.4.4 SEFI

SEFI is an anomalous behavior observed in complex devices. It can be the result of the upset of some registers or latches that are used in the configuration of the working modes of these complex devices.

Koga [36] was among the first to define SEFI from observations in SDRAM, EEPROMS, and microprocessors. Dodd [37] defines SEFI as a triggering of an IC test mode, a reset mode or some other mode that causes the IC to temporarily lose its functionality.

SEFIs were reported in flash nonvolatile memories, SDRAM, SRAM based FPGA, microprocessors, and microcontrollers.

2.4.4.1 SEFI in Flash Memories

Bagatin et al. [38] studied 1 Gb Nand Flash and could identify different SEFIs. They classify SEFI in different ways:

- SEFI that gives a full page or block read-errors and SEFI that slows down or disables the capability of the device to program and/or to erase.
- SEFI can be also classified depending on the way the functionality was restored after the event: a power cycle can be necessary or a reset command is sufficient, or a spontaneous recovery is observed.
- Supply current increase is also observed, and nominal value is restored by reset or by power cycle. In this latter case, SEFI is difficult to separate from SEL.

2.4.4.2 SEFI in FPGA

In SRAM based Field Programmable Arrays (FPGA), such as Virtex, George [39] expected to find SEFI due to upsets in particular circuits that involve Power-on-reset (POR), failures in the JTAG or Select-Map communications port or others.

2.4.4.3 SEFI in SDRAM

In SDRAM, the block diagram shows that mode registers, address registers, refresh counter, are used to control the device. A SEU in one of these registers may modify the refresh period, or induce other sorts of SEFI.

SEFI is a complex effect that needs a special treatment in a system. Fortunately, the cross-section is generally very low.

2.4.5 Single Event Latch-Up

SEL is related to the activity of parasitic structures in bulk CMOS technology. Some PNPN structures can switch from a high impedance state to a low impedance one. Latch-up can be induced also by electrical transients or by high dose rate X-ray

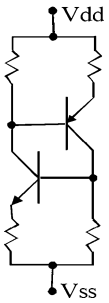


Fig. 2.11 Two bipolar transistors equivalent structure of a PNPN parasitic structure found in bulk CMOS technology

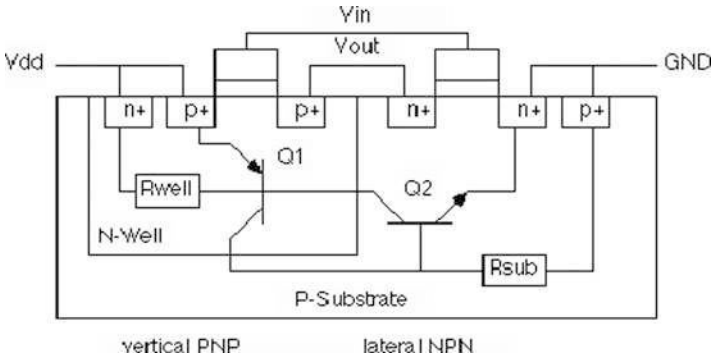


Fig. 2.12 PNPN parasitic structure in an N-Well bulk CMOS Inverter structure

pulse. The possibility of a PNPN structure to switch depends upon the following factors:

- Bias of the structure.
- Activity of the structure.
- Triggering energy delivered by the recoil.

The PNPN structure can be described by two cross-coupled bipolar transistors (one PNP and one NPN) as indicated in Fig. 2.11. Resistors are included in series with emitters and between base and emitters. In some equivalent schematics, resistors in series with collectors can be included.

The relation with the CMOS structure of an inverter in a CMOS bulk N Well Technology is given in Fig. 2.12 (after Puchner [40]).

The N-Well/P-Substrate junction is the base collector junction of both PNP and NPN junction. The emitter-base junctions are N+Source-PSubstrate and P+source-NWell junctions. The current gain of a bipolar transistor is related to the base width and the injection efficiency of the emitter-base junction. The base width varies with design rules and the position of the source relative to the Well-Substrate junction.

An ideal PNPN structure is active if the gain product of the NPN and PNP transistors is $\beta 1 \times \beta 2 > 1$, where $\beta 1$ and $\beta 2$ are the current gains Ic/Ib of the bipolar transistors.

Other important parameters that influence the sensitivity are the resistivity and depth of the well and the resistivity of the substrate. To reduce the base-emitter resistor a thin epitaxial layer on a high doped substrate can be used. Multiple substrate and well contacts also reduce the base-emitter resistors.

2.4.5.1 Latch-Up Triggering Mechanisms

The triggering mechanism can be understood by considering the equivalent scheme with cross-coupled NPN and PNP bipolar transistors.

When active the PNPN structure has two stable states: Off-state where both bipolar transistors are Off- and On-state where both bipolar transistors are On.

The minimum voltage and current needed to sustain the On-state are called respectively holding current and holding voltage.

The holding voltage must be compared to the power supply voltage: When the PNPN structure is On, both base-emitter junctions are On and the collector-base junctions are also On (saturated structure). So the minimum holding voltage between Vdd and Vss is about 0.6 V. In practice, voltage drops in series resistors increases this minimum value that may reach 0.8–1 V. In most advanced technology nodes, this holding voltage is close or greater than the power supply value. So the reduction of power supply voltage limits the occurrence of SEL (Fig. 2.13).

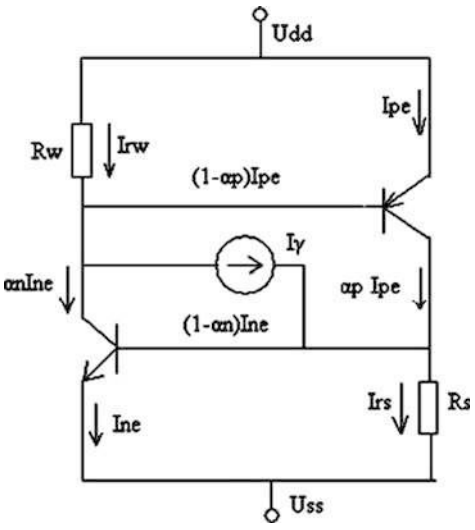


Fig. 2.13 PNPN structure trigger: The ion track crosses the N-Well/P-Substrate junction: The current is amplified in the positive feedback structure. After Useinov [41]

2.4.5.2 Design Rules Effects on SEL Sensitivity

The reduction of the base width increases the current gain and the cutoff frequency of the bipolar transistors. The junction capacitance decreases so the charge needed to bias the junction at +0.6 V is reduced. These parameters enhance the sensitivity because the trigger energy is reduced and the structure can switch with very short duration transients. But, as indicated above, at each new technology generation, the power supply decreases, the space-charge layer width and the free carrier lifetime decreases, and so the collected charge is reduced.

So a global compensation exists and SEL occurrence is difficult to predict without a detailed knowledge of the topology of the design.

SEL must be imperatively avoided by the designers. Several ways exists for eliminating it. The more drastic one is to use SOI technology that eliminates the PNPN structure. In bulk CMOS, as previously discussed, a thin epitaxial layer on a heavily doped substrate is efficient to reduce the shunt resistors in parallel with base-emitter junctions, and then a strong increase of a triggering current is needed.

SEL has been widely studied in the literature mainly for heavy ions [41, 42], but also for neutrons [43, 44]. For atmospheric neutrons-induced SEL, Dodd [43] has clearly shown that the sensitivity increases with temperature and power supply voltage. FIT values from 10 FIT to more than 300 FIT/Mb have been measured in SRAMs.

2.5 Conclusion

At ground level and in the atmosphere, SEEs are mainly due to neutrons and to alpha particles emitted by radioactive impurities. The recoils, created by neutron interaction with the nucleus of the different materials used to build the components, ionize the semiconductor and charges are collected at sensitive nodes. Soft errors (recoverable) and hard errors (non recoverable) can be induced in semiconductor devices. The basic mechanisms are rather well known, but a lot of work is still needed to accurately predict the sensitivity of the devices. For successive technology nodes, the relative importance of different effects varies. SEE prediction is a multidisciplinary task that involves nuclear and solid state physics, simulation tools, semiconductor technology, devices, and system architecture.

References

1. JESD89A: Measurement and reporting of alpha particle and terrestrial cosmic ray-induced soft errors in semiconductor devices Oct 2006, http://www.jedec.org.
2. P. Goldhagen, M. Reginato, T. Kniss, J.W. Wilson, R.C. Singleteny, I.W. Jones, and W. Van Seveninck, “Measurement of the energy spectrum of cosmic-ray induced neutrons aboard an ER-2 high altitude airplane”, Nucl. Instrum. Methods Phys. Res. A, vol. 476, pp. 42–51, 2002.

3. P. Goldhagen, “Cosmic-ray neutrons on the ground and in the atmosphere”, MRS Bulletin, pp. 131–135, Feb. 2003.
4. F. Lei, A. Hands, S. Clucas, C. Dyer, and P. Truscott, “Improvement to and validations of the QinetiQ atmospheric radiation model (QARM)”, IEEE Trans. Nucl. Sci., vol. 53, no. 4, p1851, Aug. 2006, http://qarm.space.qinetiq.com/description/qarm_model.html.
5. M.B. Chadwick, and E. Normand, “Use of new ENDF/B-VI proton and neutron cross sections for single event upset calculations”, IEEE Trans. Nucl. Sci., vol. 46, no. 6, p1386, 1999.
6. M.B. Chadwick, P.G. Young, R.E. MacFarlane, P. Moller, G.M. Hale, R.C. Little, A.J. Koning, and S. Chiba, “LA150 documentation of cross sections, heating, and damage,” Los Alamos National Laboratory report LA-UR-99-1222 (1999).
7. F. Wrobel, “Nucleon induced recoil ions in microelectronics” LPES-CRESA, University of Nice, Sophia-Antipolis, France International Conference on Nuclear Data for Science and Technology 2007.
8. Y. Tukamoto, Y. Watanabe, and H. Nakashima, “Analysis of cosmic ray neutron-induced single-event phenomena” Proceedings of the 2002 Symp. on Nuclear Data, Nov. 21–22, 2002, JAERI, Tokai, Japan, JAERI-Conf. 2003–2006, p. 265, (2003).
9. K. Niita et al, JQMD code, JAERI-Data/Code 99-042 (1999).
10. S. Furihata, Nucl. Instrum. Methods Phys. Res. B, vol. 171, p251, 2000.
11. E. Normand, “Extensions of the burst generation ate method for wider application to proton/neutron-induced single event effects”, IEEE Trans. Nucl. Sci., vol. 45, no. 6, p2904, Dec. 1998.
12. J.F. Ziegler, SRIM 2008, www.srim.org.
13. H. Paul, Stopping power for light ions, MSTAR, www.exphys.uni-linz.ac.at/stopping/ http://www-nds.ipen.br/stoppinggraphs/ www.exphys.uni-linz.ac.at/stopping.
14. M. Murat, A. Akkerman, and J. Barak, “Electron and ion tracks in silicon: Spatial and temporal evolution”, IEEE Trans. Nucl. Sci., vol. 55, no. 6, p3046, Dec. 2008.
15. A.S. Kobayashi, A.L. Sternberg, L.W. Massengill, Ronald D. Schrimpf, and R.A. Weller, “Spatial and temporal characteristics of energy deposition by protons and alpha particles in silicon”, IEEE Trans. Nucl. Sci., vol. 51, no. 6, p3312, Dec. 2004.
16. X.W. Zhu, “Charge deposition modelling of thermal neutron products in fast submicron MOS devices”, IEEE Trans. Nucl. Sci., vol. 46, 1996.
17. F.W. Sexton, “Destructive single-event effects in semiconductor devices and ICs”, IEEE Trans. Nucl. Sci. vol. 50, no. 3, Part 3, pp. 603–621, June 2003.
18. G. Hubert, J.-M. Palau, K. Castellani-Coulie, M.-C. Calvet, S. Fourtine, “Detailed analysis of secondary ions’ effect for the calculation of neutron-induced SER in SRAMs”, IEEE Trans. Nucl. Sci., vol. 48, no 6, pp. 1953–1959, Dec. 2001.
19. P.E. Dodd, and F.W. Sexton, “Critical charge concepts for CMOS SRAMs”, IEEE Trans. Nucl. Sci., vol. 42, no. 6, p1764, Dec. 1995.
20. K. Yamagushi et al, “3-D device modeling for SRAM soft-error immunity and tolerance analysis”, IEEE Trans. Electron Devices. vol. 51, no.3, p378, Mar. 2004.
21. P. Jain, and V. Zhu “Judicious choice of waveform parameters and accurate estimation of critical charge for logic SER”.
22. D. Munteanu, and J.-L. Autran, “Modeling and simulation of single-event effects in digital devices and ICs”, IEEE Trans. Nucl. Sci., vol. 55, no. 4, pp. 1854–1878, Aug. 2008.
23. M.A. Bajura et al, “Models and algorithmic limits for an ECC-based approach to hardening sub-100-nm SRAMs”, IEEE Trans. Nucl. Sci., vol. 54, no. 4, pp. 935, Aug. 2007.
24. P.E. Dodd, “Device simulation of charge collection and single-event upset”, IEEE Trans. Nucl. Sci., vol. 43, pp. 561–575, 1996.
25. P.E. Dodd et al, “Production and propagation of single-event transients in high-speed digital logic ICs”, IEEE Trans. Nucl. Sci., vol. 51, no. 6, pp. 3278–3284, Dec. 2004.
26. M. Hane, H. Nakamura, K. Tanaka, K. Watanabe, Y. Tosaka, “Soft error rate simulation considering neutron-induced single event transient from transistor to LSI-chip level”, 2008 Intern. Conf. on Sim. of Semiconductor Processes and Devices (SISPAD 2008).

27. S. Buchner et al, “Comparison of error rates in combinational and sequential logic”, IEEE Trans. Nucl. Sci., vol. 44, no. 6, pp. 2209–2216, Dec. 1997.
28. R.C. Baumann, “Radiation-induced soft errors in advanced semiconductor technologies”, IEEE Trans. Device Mater. Reliab., vol. 5, no. 3, pp. 305–316, Sep. 2005.
29. C. Shuming, L. Bin, L. Biwei, and L. Zheng, “Temperature dependence of digital SET pulse width in bulk and SOI technologies”, IEEE Trans. Nucl. Sci., vol. 55, no. 6, pp. 2914, Dec. 2008.
30. P. Bai et al, “A 65 nm logic technology featuring 35 nm Gate lengths, enhanced channel strain, 8 Cu Interconnect layers, low k ILD and 0.57 μm^2 SRAM Cell” IEDM 2004.
31. P.E. Dodd, “Physics-based simulation of single-event effects”, IEEE Trans. Device Mater. Reliab., vol. 5, no. 3, pp. 343–357, Sep. 2005.
32. Leif Z. Scheick, Steven M. Guertin, and Gary M. Swift, “Analysis of radiation effects on individual DRAM cells”, IEEE Trans. Nucl. Sci., vol. 47, no. 6, p2534, Dec. 2000.
33. T. O’Gorman, “The effect of cosmic rays on the soft error rate of a DRAM at ground level”, IEEE Trans. Electron Devices, vol. 41, no. 4, p553, Apr. 1994.
34. Lloyd W. Massengill, “Cosmic and terrestrial single-event radiation effects in dynamic random access memories”, IEEE Trans. Nucl. Sci., vol. 43, no. 2, p516, Apr. 1996.
35. E. Takeda, D. Hisamoto, T. Toyabe, “A new soft-error phenomenon in VLSIs: The alpha-particle-induced source/drain penetration (ALPEN) effect.” International Reliability Physics Symposium 1988. 26th Annual Proceedings, 12–14 April 1988 Page(s):109–112.
36. R. Koga, S.H. Penzin, K.B. Crawford, W.R., Crain, “Single event functional interrupt (SEFI) sensitivity in microcircuits” Radiation and Its Effects on Components and Systems, 1997. RADECS 97. Fourth European Conference on 15–19 Sept. 1997 Page(s):311–318.
37. P.E. Dodd, and L.W. Massengill, “Basic mechanisms and modeling of single-event upset in digital microelectronics,” IEEE Trans. Nucl. Sci., vol. 50, no. 3, pp. 583–602, Jun. 2003.
38. M. Bagatin et al, “Key contributions to the cross section of NAND flash memories irradiated with heavy ions”, IEEE Trans. Nucl. Sci., vol. 55, no. 6, p3302, Dec. 2008.
39. J. George, R. Koga, G. Swift, G. Allen, C. Carmichael, and C.W. Tseng, Single Event Upsets in Xilinx Virtex-4 FPGA Devices Radiation Effects Data Workshop, 2006 IEEE, July 2006 pp. 109–114.
40. H. Puchner, R. Kapre, S. Sharifzadeh, J. Majjiga, R. Chao, D. Radaelli, and S. Wong, “Elimination of single event latchup in 90 nm sram technologies” 4th Annual IEEE IRPS Proceedings, 2006, March 2006 Page(s):721–722.
41. R.G. Useinov, “Analytical model of radiation induced or single event latchup in CMOS integrated circuits”, IEEE Trans. Nucl. Sci., vol. 53, no. 4, p1834, Aug. 2006.
42. F. Bruguier, and J.-M. Palau, “Single particle-induced latchup”, IEEE Trans. Nucl. Sci., vol. 43, no. 2, pp. 522–532.
43. P.E. Dodd, M.R. Shaneyfelt, J.R. Schwank, and G.L. Hash, “Neutron-induced soft errors, latchup, and comparison of SER test methods for SRAM technologies”, IEDM Tech. Dig., pp. 333–336, 2002.
44. J. Tausch, D. Sleeter, D. Radaelli, and H. Puchner, “Neutron Induced Micro SEL Events in COTS SRAM Devices” 2007 IEEE Radiation Effects Data Workshop, pp. 185–188.
45. K.P. Rodbell, D.F. Heidel, H.H.K. Tang, M.S. Gordon, P. Oldiges, and C.E. Murray, “Low-energy proton-induced single-event-upsets in 65 nm node, silicon-on-insulator, latches and memory cells”, IEEE Trans. Nucl. Sci. vol. 54, no. 6, p2474, Dec. 2007.
46. David F. Heidel et al, “Low energy proton single-event-upset test results on 65 nm SOI SRAM”, IEEE Trans. Nucl. Sci. vol. 55, no. 6, p3394, Dec. 2008.
47. S. Furihata, and T. Nakamura, J. Nucl. Sci. Technol., Suppl. 2, 758, 2002.
48. J.F. Ziegler, J.P. Biersack, U. Littmark, The Stopping and Range of Ion in Solids, Pergamon Press, 1985.

Chapter 3
JEDEC Standards on Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray Induced Soft Errors

Charles Slayman

While the history of soft errors in commercial semiconductor devices spans over three decades, it has only been relatively recently that specifications have been created to standardize the characterization of the effects of alpha particles and neutrons on ICs. Some of the first standards developed for devices used in commercial applications come from one of the premier semiconductor industry standards body, JEDEC, formerly known as Joint Electron Device Engineering Council. The JEDEC JESD89 standards are now widely referenced in most technical publications on soft errors in commercial ICs. This chapter gives an overview of the JEDEC JESD89 series of standards, describes the technical background for their development and details the areas for future improvement.

3.1 Introduction

The Joint Electron Device Engineering Council (JEDEC) was formed in 1958 as a part of the Electronic Industries Association (EIA). In 1999, JEDEC was incorporated as an independent association under the name, *JEDEC Solid State Technology Association* (thereby abandoning any reference to the original acronym). This new organization then became a recognized sector of EIA and maintains that relationship today [1].

JEDEC is the leading developer of standards for the solid-state industry. Almost 3,300 participants, appointed by some 295 companies work together in 50 JEDEC committees to meet the needs of every segment of the industry, manufacturers and consumers alike. The publications and standards that they generate are accepted throughout the world. All JEDEC standards are available online, at no charge [2].

C. Slayman (✉)
Ops La Carte, 990 Richard Ave. Suite 101, Santa Clara, CA 95050, USA
e-mail: charlies@opsalacarte.com

M. Nicolaidis (ed.), *Soft Errors in Modern Electronic Systems*,
Frontiers in Electronic Testing 41, DOI 10.1007/978-1-4419-6993-4_3,
© Springer Science+Business Media, LLC 2011

3.1.1 Purpose of the JESD89 Series Standards

While the history of soft errors commerical in microelectronics dates back to 1979 [3–5], it is somewhat surprising that as recently as 2000, no test standards existed to make the characterization of soft errors uniform in a terrestrial environment, even though a significant amount of knowledge had been gained in the military, aerospace and space environments. This situation led to considerable confusion among commercial design and system engineers in terms of what error rates they could expect. To that end, a task group of commercial and aerospace semiconductor experts was assembled at the request of NASA to determine the best way to standardize SER testing. It was decided that JEDEC would be the appropriate technical body for publication and maintenance of the standards that were primarily targeted for commercial devices.

The purpose of the JESD89 standards is to provide a procedure to characterize the soft error rate of ICs in a terrestrial environment. The sources of errors include radiation from (1) alpha particles within the IC package as well as the IC itself and (2) high energy and thermal neutrons resulting from cosmic rays impinging on the earth's atmosphere. JESD89A *Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices* is the overarching standard that covers all three (alpha, thermal neutron and high energy neutron) sources of soft errors in detail. JESD89-1A *Method for Real-Time Soft Error Rate* focuses on the technique of using a large number of devices over an extended period of time to measure soft errors. JESD89-2A *Test Method for Alpha Source-Accelerated Soft Error* Rate deals with the use of alpha particle sources to characterize the soft error rate of unpackaged or de-lidded devices. JESD89-3A *Test Method* for *Beam-Accelerated Soft Error Rate* details the error rate characterization of devices under neutron or proton beam irradiation. A comprehensive characterization of alpha, high-energy neutron and thermal neutron effects are required in order to fully characterize the soft-error rate of an IC. The most recent version of all four standards can be found at www.jedec.org/download/default.cfm by using JESD89 in the Search by Document Number box.

3.1.2 Terms and Definitions

The JESD89 standards give definitions of many of the terms commonly used in soft error measurement and characterization. These can be found in JESD89A Sect. 2, JESD89-1A Sect. 3, JESD89-2A Sect. 3 and JESD89-3A Sect. 4. The following is a list of some of the key terms.

Alpha activity the number of alpha particles that decay in an alpha source per unit time. The SI unit is Becquerel (Bq) which is one disintegration per second. 1 curie $= 3.7 \times 10^{10}$ Bq.

Critical charge (Qc) the minimum amount of collected charge that will cause the node of a circuit to change state.

Differential flux the rate of particles incident on a surface per unit energy per unit area. Example of units – cm^{-2} s^{-1} eV^{-1}.

DUT device under test.

Fluence the number of radiant energy particles emitted from (or incident on) a surface over a given period of time divided by the surface area. Example of units – cm^{-2}.

Flux density the rate of radiant energy particles emitted from (or incident on) a surface per unit area. Example of units – cm^{-2} s^{-1}.

Golden part a sample used to monitor the consistency of a radiation beam and test setup.

Hard error an irreversible change in the operation of a device that is typically associated with permanent damage to one or more elements of the device (e.g. gate oxide rupture destructive latch-up events, interconnect opens or shorts, etc.)

Multiple-bit upset (MBU) a multi-cell upset (MCU) in which two or more error bits occur in the same word.

Multiple-cell upset (MCU) a single event that induces several bits in a memory array to fail at the same time.

Single-event burnout (SEB) an event in which a single energetic particle strike induces a localized high current state that results in catastrophic failure.

Single-event effect (SEE) any measureable or observable change in the state or performance of a device resulting from an energetic particle strike.

Single-event functional interrupt (SEFI) an SEE that causes a device reset lock-up or otherwise malfunction, but does not require power cycling to restore operability.

Single-event hard error (SHE) an irreversible change in the operation of a device due to an SEE.

Single-event latch-up (SEL) an abnormally high-current state in a device or circuit of a device caused by the passage of an energetic particle and leading to localized latch-up in that region.

Single-event transient (SET) a momentary voltage or current spike at a node of a circuit caused by the passage of a single energetic particle.

Single-event upset (SEU) the change in state of a circuit caused by a single energetic particle.

Soft error (SE) an erroneous output signal from a latch or memory cell that can be corrected by performing one or more normal device functions.

Power cycle soft error (PCSE) a soft error that cannot be corrected by simple re-write operations but requires the removal and restoration of power to the device (e.g. non-destructive latch-up).

Static soft error a soft error that is not corrected by repeated reading but can be corrected by rewriting without the removal of power.

Transient soft error a soft error that can be corrected by repeated reading without rewriting or the removal of power.

3.1.3 Devices Covered by the Standards

JESD89A is relatively general and can be applied to both memory and logic components. JESD89-1A, 2A and 3A focus on solid state volatile memory arrays (e.g. DRAM and SRAM) and bistable logic elements (e.g. flip-flops) for errors that require no more than a re-read or re-write/read to determine if the error is noise induced, soft error or hard error.

3.1.4 Reporting Requirements

An important part of any test standard is the requirement of how the data is presented. Each of the JESD89 standards lays out a detailed list of what information is required and must be included in the final report. Alpha particle and neutron events can depend on many external environmental and internal semiconductor parameters, the physics of which might be only partially understood. For example, generation and attenuation of alpha particles in an IC package or generation and decay of thermal neutrons from the high-energy neutron background are difficult to predict in real field applications. The JESD89 standards attempt to model these effects, but in the end it is important to state the details of soft error measurements in the final report to allow for future interpretation and correction as knowledge of these effects improve.

The preferred unit of reporting is FIT per device or a normalized FIT per bit. A FIT is defined as 1 failure (or more properly for soft errors, 1 event) per 10^9 h. If there are different types of soft error events observed, then they should be reported separately. For example, it is not appropriate to combine soft error latch-up events with cell upset events.

3.2 Accelerated Alpha Particle SER Testing (JESD89A Sect. 4 and JESD89-2A)

3.2.1 Alpha Particle Energy Spectra and Emissivity (JESD89A Annex D)

The energy spectra of alpha particles emitted from radioactive decay is a function of the isotope that contaminates the package or device. The activity of an isotope is proportional to its natural abundance and inversely related to its half-life (the time required for half the original population of atoms to decay). $_{238}$U, $_{235}$U and $_{232}$Th have the highest activities of naturally occurring radioactive contamination. If the half-life of the daughter products is less than the half-life of the parent isotope, then these will also contribute to the alpha energy spectra.

Figure 3.1 shows the alpha energy spectra of $_{238}$U, $_{235}$U and $_{232}$Th.

Since alpha particles are charged, they lose energy as they penetrate through the package and device. So the discrete energy spectra in Fig. 3.1 is broadened and attenuated in thick films. Figure 3.2 shows the energy spectra of a thick $_{232}$Th film.

Alpha particles have a limited range due to their energy loss. Elastic coulomb scattering of the alpha particle with electrons generates electron-hole pairs. In silicon, each electron-hole pair generated represents an average energy loss of 3.6 eV. Figure 3.3 shows the linear energy transfer measured in charge transfer

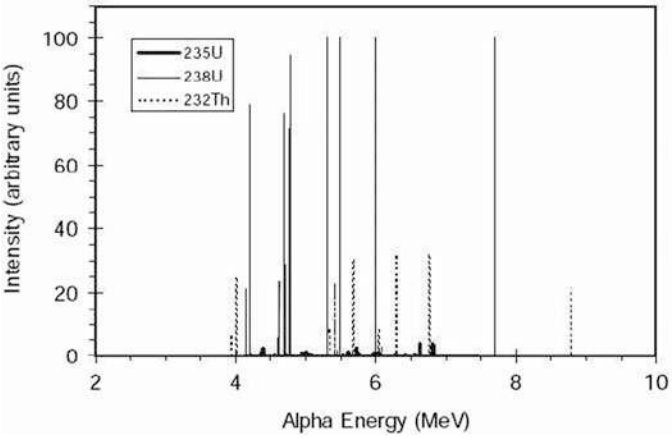


Fig. 3.1 Emission spectra from thin-film $_{238}$U, $_{235}$U and $_{232}$Th. (Reprinted from JESD89A p 75)

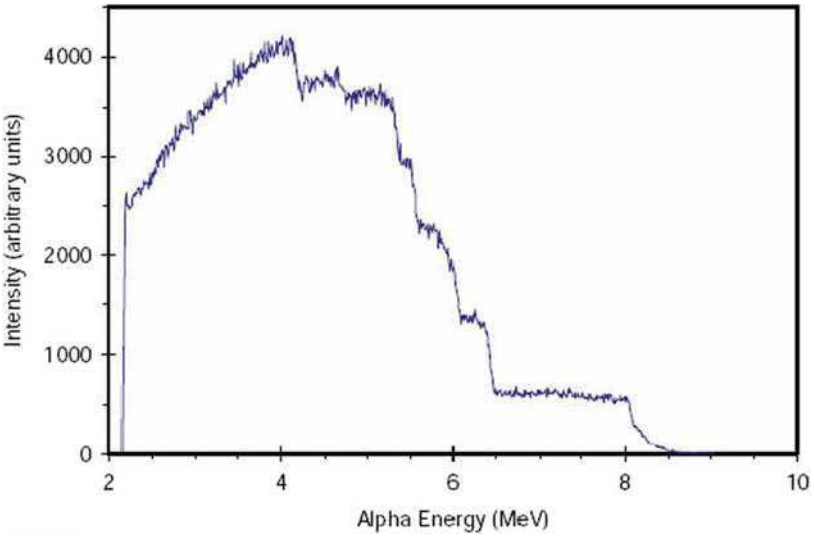


Fig. 3.2 Emission spectrum of a thick-film $_{232}$Th source. (Reprinted from JESD89A p 76)

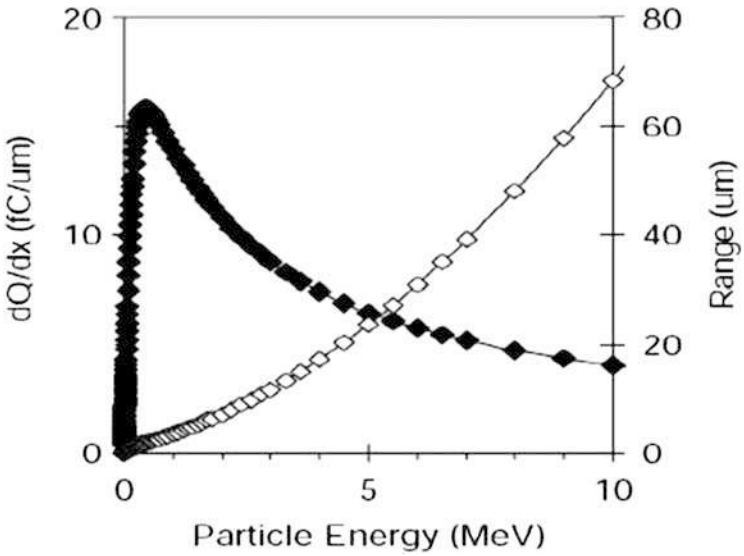


Fig. 3.3 Charge transfer (*solid boxes*) and range (*open boxes*) of alpha particles in silicon. (Reprinted from JESD89A p 77)

Table 3.1 Alpha emissivity of various electronic materials. (Reprinted from JESD89A, p. 78)

Material	Alpha emissivity (cm^{-2} h^{-1})
Fully processed wafers	$<4 \times 10^{-4}$
30 μm thick Cu interconnect	$<3 \times 10^{-4}$
20 μm thick AlCu interconnect	$<3 \times 10^{-4}$
Mold compound	$<5 \times 10^{-4}$ to $<2.4 \times 10^{-2}$
Flip-chip underfill	$<7 \times 10^{-4}$ to $<4 \times 10^{-3}$
Eutectic Pb-based solder	$<9 \times 10^{-4}$ to <7.2

per unit distance (fC/μm) in silicon as a function of alpha energy. It is maximum around 1 MeV and drops off at higher alpha energies. As shown in Fig. 3.3, the range of an alpha particle in silicon is also a function of its energy. A 1 MeV alpha particle can only penetrate a few μm, whereas a 5 MeV particle can penetrate over 20 μm.

A useful guideline is given in JESD89A Annex D and shown in Table 3.1 below as to what range of alpha emissivity to expect from various materials used in IC technology. It should be noted that the examples are for a well controlled and monitored manufacturing environment. If contamination occurs from uncontrolled sources, the emissivities can increase by orders of magnitude. Emissivities below

0.001 counts/(cm^2 h) are approaching the particle counter detection limit and require well-controlled laboratory techniques.

3.2.2 Alpha Source Selection (JESD89A Sect. 5.4.1 and JESD89-2A Sect. 4.2.2.1)

Selection of the alpha source species should depend upon the appropriate impurity expected in the package and device. Americium (Am) sources have similar spectra to $_{210}$Po that would be found in flip-chip solder bumps using lead. Thorium (Th) sources have similar spectra to uranium impurities found in moulding compound. Thorium itself can also be an impurity in moulding compounds.

As noted above, thin film or metal foil sources have discrete energy spectra. This is a representative spectrum if the real life source of alpha particles in the product is coming from a thin layer at the die surface. However, this is not the typical source of alpha particles in a device where the moulding compound is responsible for the alpha particles. Solid sources that are physically thicker will broaden the alpha spectra. This is representative of alpha particles being emitted throughout the bulk of the device package. These factors should be taken into account along with the source species when selecting the proper source for accelerated alpha particle testing.

3.2.3 Packaging/Sample Preparation (JESD89A Sect. 5.3 and 5.4.5, JESD89-2A Sect. 4.4)

Since the penetration depth of alpha particles is very short, accelerated alpha testing must be done with bare die in either wafer form with special probe cards or in specially modified dual-in-line (CERDIP) or pin-grid array (CERPGA) packages. Alpha source to die spacing must be minimized to prevent attenuation of alpha flux from air: a spacing of less than 1 mm is recommended.

If a protective overcoat and alpha blocking layer (e.g. polyimide) is used in the product, it should remain intact for the testing. If it is removed, an estimation of its attenuation factor must be noted in the final report in order to translate between an accelerated alpha particle SER and the normal alpha particle SER.

3.2.4 Extrapolating Accelerated Failure Rate to Field Use Conditions (JESD89A Sect. 5.6.4)

Extrapolation from the accelerated alpha soft error rate (R_{acc}) to the normal alpha particle soft error rate expected in field (R_{α}) use is given by

$$R_\alpha = R_{\text{acc}} \frac{\phi_{\text{pkg}} F_{\text{geopkg}} F_{\text{shieldpkg}}}{\phi_{\text{dut}} F_{\text{geodut}} F_{\text{shielddut}}} \tag{3.1}$$

where ϕ_{pkg} is the background alpha flux from the actual product package and device impurities, ϕ_{dut} is the flux from the alpha particle source on the device under test (DUT). F_{geopkg} and F_{geodut} are geometrical factors to account the reduction of flux due spacing from the source. If the source to device spacing rule of <1 mm is followed, then $F_{\text{geopkg}} = F_{\text{geodut}}$. $F_{\text{shieldpkg}}$ and F_{shieldut} account for the shielding layers (e.g. polyimide, inter-level dielectrics and metal interconnects) between the source and device. If the device used for accelerated testing is the same as the product device (i.e. same top layers and coating), the $F_{\text{shieldpkg}} = F_{\text{shieldut}}$. Otherwise, difference in alpha particle attenuation between the test device and the product must be calculated.

3.2.5 *Limitations and Strengths of the Accelerated Alpha Particle Test*

The strong point of accelerated alpha particle SER testing is that sources are readily available and test times can be relatively short to collect a statistically significant number of errors. This enables rapid determination of which circuits are sensitive to alpha upset (independent of neutron effects) and allows for added design cycles to improve alpha particle SER performance if required.

One of the drawbacks of the test is that it can be complicated to produce the actual energy spectra at the surface of the device due to the shielding and attenuation that occurs within the package material itself. (Solid alpha sources are preferred over foil sources for this reason.) The self-shielding of the package material will tend to shift the peak of the energy deposition (Bragg peak) within the device. Some devices might be more sensitive at the surface, and other devices might be more sensitive deeper into the silicon. So shifting of the Bragg peak into the sensitive area by added shielding can actually cause an increase in the αSER. The only way to determine this effect would be to do multiple accelerated alpha particle experiments with different amounts of shielding.

For flip-chip packaged devices, the standard does not cover how to calculate the effects of localized alpha sources coming from the lead bumps. These geometrical effects are very different than radiation from a uniform alpha source. In addition, there are severe limitations on the speed the device can be tested if it is not in a flip-chip package. So predicting real life SER rates from accelerated alpha SER in flip-chip devices is less accurate than more conventional devices packaged in moulded compounds.

3.3 Accelerated High Energy Neutron SER Testing (JESD89A Sect. 6 and JESD89-3A)

3.3.1 Terrestrial High Energy Neutron Flux and Spectra (JESD89A Sect. 6.6.2.4)

Interaction of high-energy galactic cosmic rays with atomic nuclei in the earth's upper atmosphere lead to a cascade of every kind of fundamental particle. High energy neutrons (>1 MeV) are the dominant particles that can (1) reach terrestrial levels (0 to ~10,000 ft above sea level), (2) penetrate an IC package and (3) induce electron-hole pair generation in sufficient quantities to upset a circuit.

The actual neutron flux depends on several factors:

- Geographical Location (latitude and longitude) due to the influence of the earth's magnetic fields on the charged particles that lead to the generation of free neutrons.
- Altitude and Barometric Pressure because the earth's atmosphere attenuates the neutron flux.
- Solar Activity, which has an effect on the earth's magnetic field.

The neutron flux is also highly energy dependent – $\phi(E)$. The differential flux is the neutron flux per unit of energy – $d\phi(E)/dE$. JESD89A uses New York City as

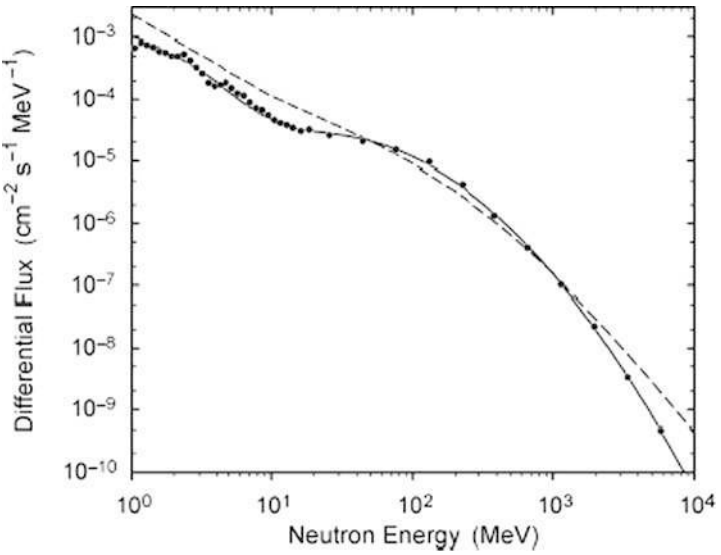


Fig. 3.4 Differential high-energy neutron spectra at New York City. Data points are the reference spectrum provided in Table A.2.A of JESD89A and the solid line is from (3.2). The dashed line is from the original version of JESD89 (Aug. 2001). (Reprinted from JESD89A p 58 and [8] ©2004 IEEE)

the reference neutron spectrum. Table A.2.A of JESD89A Annex A tabulates reference spectrum from 1 MeV to 150 GeV. An analytic expression for the differential flux at New York City is also provided to fit the reference spectrum.

$$\frac{\mathrm{d}\phi_{\mathrm{NYC}}(E)}{\mathrm{d}E} = 1.006 \times 10^{-6} \mathrm{e}^{-0.35(\ln(E))^2 + 2.1451 \ln(E)} + 1.011 \times 10^{-3} \mathrm{e}^{-0.4106(\ln(E))^2 - 0.667 \ln(E)} \tag{3.2}$$

Figure 3.4 shows the outdoor differential high energy neutron spectra referenced to New York City during a period of mid-level solar activity. Note that both axes of the figure are logarithmic scale. The differential flux of low energy neutrons is orders of magnitude higher than higher energy neutrons. This dominance of low energy neutrons can be misleading because the higher energies (100 MeV–10 GeV) are compressed in comparison to the lower energies (1–100 MeV) due to the logarithmic nature of the plot. In addition, the higher energy neutrons generate more charge when they strike a silicon nucleus, and therefore play a more significant role in soft errors.

3.3.2 *Scaling of Reference Spectrum to Other Locations and Conditions (JESD89A Annex A.3)*

The shape of the outdoor neutron spectra above a few MeV does not change significantly as a function of barometric pressure, altitude, geomagnetic rigidity or solar activity. This makes it straightforward to calculate the outdoor neutron flux at any terrestrial location. Analytic equations are provided in JESD89A Annex A.3 to account for the effects of altitude, geomagnetic cutoff and solar activity. The differential flux at any given location ($\mathrm{d}\phi_{\mathrm{loc}}(E)/\mathrm{d}E$) is

$$\frac{\mathrm{d}\phi_{\mathrm{loc}}(E)}{\mathrm{d}E} = \frac{\mathrm{d}\phi_{\mathrm{NYC}}}{\mathrm{d}E} F_{\mathrm{A}}(d) F_{\mathrm{B}}(R_{\mathrm{c}}, I, d) \tag{3.3}$$

where d is the atmospheric depth, R_{c} is the vertical geomagnetic cutoff rigidity and I is the relative count rate of a neutron monitor measuring solar activity. $F_{\mathrm{A}}(d)$ is a function describing the principle dependence on altitude and barometric pressure and $F_{\mathrm{B}}(R,I,d)$ is the function depending on geomagnetic location and solar activity. The detailed dependence of F_{A} and F_{B} can be found in JESD89A Annex A.3. All of these effects are captured in the flux calculation tool that can be found at http://www.seutest.com/cgi-bin/FluxCalculator.cgi.

3.3.3 *Test Facilities (JESD89A Sect. 6.2)*

Three different types of facilities are appropriate for accelerated high energy neutron testing: (1) spallation neutron sources, (2) quasi-mono-energetic neutron

sources and (3) mono-energetic proton sources. Spallation neutron sources generate neutrons over a wide spectrum similar to terrestrial neutrons. The advantage is that one beam can act as an extremely intense source of neutrons with the correct energy spectrum.

Since there are a limited number of these sources worldwide, mono-energetic neutron and proton sources provide a viable alternative. Above 50 MeV, protons interaction with matter is essentially the same as neutrons with the same energy. Since proton beams are easier to produce and control, mono-energetic protons >50 MeV can be substituted for neutrons. Use of quasi-mono-energetic neutron sources requires unfolding techniques (i.e. deconvolving the SEU cross-section as a function of energy) that are not firmly established in JESD89A at this point. An added advantage of the mono-energetic and quasi-mono-energetic sources is that they allow more insight into the parts of the neutron energy spectra that are contributing to a devices soft error rate.

An up to date list of all the neutron and proton facilities can be found at http://www.seutest.com/mwiki/index.php?title=Facilities. As new facilities come on line, they are added to the Web site.

3.3.4 Packaging/Sample Preparation and Secondary Ion Effects (JESD89-3A Sect. 5.4 and Annex A)

Since high-energy neutrons pass through IC packages virtually unattenuated, no special packaging is required. Devices should be tested in their production packages. However, neutrons are more likely to be scattered by high Z materials and generate secondary ions. With high Z materials, such as Cu used in heatsinks, it is advisable to characterize the device soft error rate with and without the heatsink (or with the heatsink in front of the device and behind the device) to quantify the interaction of the neutron beam with package. Likewise, for flip-chip packages using Pb bumps, the impact of neutron spallation will depend on whether the bumps are in front or behind the beam. In real system applications, devices are likely to be oriented in all directions. So it is useful to determine the impact of heatsink or Pb bump orientation on soft error rates.

If a high-energy proton source is used, both flux attenuation and energy dissipation of the package must be considered. Annex A of JESD89-3A gives useful tables of proton energy transfer and projected range in Al, Cu, Si, SiO_2 and epoxy using the SRIM program [6]. For example, a 90 MeV proton in epoxy (JESD89-3A Table 3) has an energy loss of 1.42 MeV/mm and projected range of 35.33 mm. A 3 mm thick epoxy layer will reduce the energy by 4.3 MeV (~5%), but the flux attenuation will be insignificant because the thickness is much less than the range. More detailed calculations of package effects can be determined using TRIM/SRIM codes [6] available at http://www.srim.org/.

JESD89-3A Sect. 5.4 makes note that finned heatsinks should be avoided due to unpredictable beam attenuation. In addition, secondary ions effects can be

minimized by spacing multiple DUTs farther apart in the beamline to allow the flux per unit area to decrease. If detailed calculations of neutron flux under various shielding conditions are required, JESD89A refers to other sources for high-energy radiation transport codes – Mcnpx [http://mcnpx.lanl.gov/], Fluka [http://pcfluka.mi.infn.it] or Geant [http:wwwasd.web.cern.ch/wwasd/geant/].

3.3.5 Beam Characteristics (JESD89A Sect. 6.5)

The facility operating the neutron or proton beam must provide expertise on several critical factors, such as the techniques to measure beam flux, temporal stability, spatial uniformity and collimation, energy spectrum and stray species. In addition, care must be taken that the devices are not in saturation due to high upset rates. If beam attenuation is required, proper techniques and materials are required to maintain the energy spectrum, spatial uniformity and spectral purity. In addition, a golden unit (i.e. a device with a previously characterized soft error rate) can be used to validate the results or determine if there are reproducibility problems.

3.3.6 Soft Error Rate for Mono-energetic Beams (JESD89A Sect. 6.6)

The nuclear interactions of protons and neutrons with the low Z materials in ICs (primarily Si and O) are the same above 50 MeV. Therefore, mono-energetic protons can be substituted for neutrons in determining the component of the soft error cross-section above 50 MeV.

The energy dependent single event cross-section is defined as

$$\sigma(E)_{\text{seu}} = \frac{N_{\text{seu}}}{\phi(E)T} \tag{3.4}$$

where N_{seu} is the number of events (not bits), $\phi(E)$ is the flux at energy E (particles/cm^2) and T is the exposure time. The soft error rate due to high energy neutrons, R_{nhe} is related to SEU cross-section, $\sigma_{\text{seu}}(E)$, by the integral

$$R_{\text{nhe}} = \int_{E_{\text{min}}}^{E_{\text{max}}} \frac{\mathrm{d}\phi(E)}{\mathrm{d}E} \sigma_{\text{seu}}(E)\mathrm{d}E \tag{3.5}$$

Emin is the cut-off energy below which no upsets occur. Emax is the maximum neutron energy. The cross-section, $\sigma_{\text{seu}}(E)$, drops off rapidly below 10 Mev and is relatively flat above 200 MeV. Therefore, a sampling of $\sigma_{\text{seu}}(E)$ between 10 and 200 MeV is sufficient to predict the soft error rate. JESD89A recommends a four-parameter Weibull function to fit the cross-section:

$$\sigma_{\text{weib-seu}} = \sigma_{\text{p/n-l}}\left(1 - \text{e}^{-\left(\frac{E-E_0}{W}\right)^S}\right) \tag{3.6}$$

where $\sigma_{\text{p/n-l}}$ is the high-energy asymptotic proton or neutron cross-section (i.e. >200 MeV), E_0 is the cut-off energy below which the cross-section is zero, W is the width parameter and S is the shape factor. In order to obtain a good fit of these four parameters, a minimum of four different energies are recommended: (1) 14 MeV mono-energetic neutron, (2) 50–60 MeV proton, (3) 90–100 MeV proton and (4) >200 MeV proton. As the cut-off energy scales down with device technology, lower mono-energetic neutron energies might be required in the future.

The soft error rate (3.5) can be approximated using a linear-log algorithm because the terrestrial differential flux ($\text{d}\phi(E)/\text{d}E$) drops by several orders of magnitude from 10 MeV to 1 GeV:

$$R_{\text{nhe}} = \sum_{j=1}^{N-1} \frac{(F_{j+1} - F_j)(E_{j+1} - E_j)}{\ln\left(\frac{F_{j+1}}{F_j}\right)} \tag{3.7}$$

where F_j is defined as

$$F_j = \sigma_{\text{seu}}(E_j)\frac{\text{d}\phi(E_j)}{\text{d}E} \tag{3.8}$$

$j = 1$ corresponds to the lowest energy point and $j + 1 = N$ is the highest (asymptotic) energy point.

3.3.7 Soft Error Rate from Broad Spectrum Neutron Beam (JESD89A 6.6.2.4)

The SEU cross-section of a device, σ_{nhe}, determined by using a broad spectrum neutron source is given by

$$\sigma_{\text{nhe}} = \frac{N_{\text{nhe}}}{\phi_{\text{spec}}T} \tag{3.9}$$

where N_{nhe} is the number of events per device during the test, ϕ_{spec} is the JESD89A specified (energy) integrated flux of neutrons (cm^{-2}) with energies >10 MeV and T is the duration of the test time (s). Note that this cross-section is not per unit energy because it is already averaged over neutron energies. >10 MeV is selected as the lower cut-off because neutrons below 10 MeV will not contribute to soft errors, and so they should not be counted. If multiple devices are in the beam during the test, then N_{nhe} is the total number of upsets divided by the number of devices. This assumes all the devices that are uniformly radiated and the error rates from device to

device are consistent. Note that this is an energy averaged cross-section since the beam is a broad energy spectrum and it is not known which neutron energies contribute to the upsets.

Using the integrated neutron flux >10 MeV at the reference New York City level of 3.6×10^{-3} cm^{-2} s^{-1} (13 cm^{-2} h^{-1}), the soft error rate of a device due to high-energy neutrons is

$$R_{\text{nhe}} = 3.6 \times 10^{-3} \sigma_{\text{nhe}} = 3.6 \times 10^{-3} \frac{N_{\text{nhe}}}{\phi_{\text{spec}} T} \tag{3.10}$$

The simplicity of the broad spectrum technique compared to the mono-energetic technique can be seen immediately by comparing (3.10) with (3.7). Equation (3.10) is valid as long as the cut-off energy, E_0, remains above 10 MeV. If scaling technology below 90 nm causes E_0 to drop below 10 MeV, then corrections will need to be made in the (energy) integration of terrestrial neutron flux.

3.3.8 Limitations and Strengths of the Accelerated High-Energy Neutron Test

The strong point of accelerated high energy neutron testing is that it allows for rapid device testing because neutron fluxes are 10^7–10^8 greater than terrestrial rates. A single facility can be used for a broad-spectrum spallation source. However, the contributions of the various segments of the neutron spectrum to the overall soft error rate will not be known. For example, knowledge of the cut-off energy (E_0) trend with device scaling is important to understand for future product designs. But E_0, cannot be determined by this technique. In addition, if the high-energy spectrum of the spallation source drops off more rapidly than the actual terrestrial neutron spectrum, then this technique will underestimate the soft error rate.

In the case of mono-energetic techniques, several facilities might be required order to obtain four different cross-sections. But this technique will give a better understanding to the energy dependence of the soft error rate. Recent studies [7] have shown relatively modest discrepancy (13–35%) between SER FIT calculated using the Los Alamos ICE House neutron beam and mono-energetic proton beam measurements and simulations. This agreement between techniques is actually quite good and confirms the validity of the various accelerated neutron and proton beam techniques.

Since the high-energy neutron spectrum is well understood as a function of geographical location, results from accelerated high energy neutron testing can be translated to any location in the world. However, these results are only applicable to the high-energy neutron contribution to a device's total soft error rate. The major drawback to this technique is the availability of facilities throughout the world.

3.4 Accelerated Thermal Neutron SER Testing

3.4.1 Background (JESD89A Sect. 7.1)

Thermal neutrons (n_{th}) are the result of high-energy neutron scattering and energy loss along with nuclear decay processes. The enhanced cross-section of thermal neutrons with $_{10}B$ nuclei present in the IC or the IC package creates energetic Li and He particles as well as gamma rays, which can lead to soft errors. The reaction is

$$n_{th} + {}_{10}B \rightarrow {}_{7}Li(0.84\,MeV) + {}_{4}He(1.47\,MeV) + \gamma(0.48\,MeV) \tag{3.11}$$

In cases where the composition of the materials used in the device are unknown (i.e. the presence of $_{10}B$ in borophosphosilicate glass or BPSG, polysilicon doping, substrate doping, ion implantation, etc.), then a test should be done to assess the impact of thermal neutrons on the soft error rate.

Less is known about the actual background flux of thermal neutrons than high-energy neutrons and alpha particles due to the interaction of neutrons with the surrounding environment. Materials present in the local environment will effect both generation and decay of thermal neutrons. Therefore, it is not straightforward to develop a thermal neutron reference flux level as is done for high-energy neutrons. And an analytical expression for thermal neutron flux, like (3.2) for the high-energy flux, is not possible. So the standard focuses on determining a relative soft error sensitivity using accelerated thermal neutron sources.

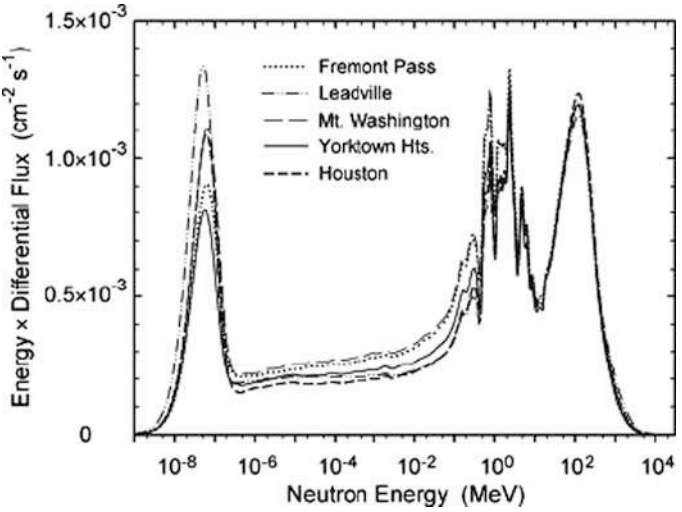


Fig. 3.5 Low and high-energy neutron spectra at five different locations. Each spectrum has been scaled to sea level and the cut-off of New York City. (Reprinted from JESD89A p 67 and [8] © 2004 IEEE)

3.4.2 Thermal Neutron Spectra (JESD89A Annex A.4)

Figure 3.5 shows the lethargy plot (energy × differential flux) of neutrons from 1 meV to 10 GeV. (Note that the range from 1 to 10,000 MeV is expanded in Fig. 3.4). The peak between 10 and 100 MeV are the thermal neutrons. The large variation in the thermal neutron flux is due to local environmental effects discussed in Sect. 3.4.1.

3.4.3 Packaging/Sample Preparation (JESD89A Sect. 7.3)

Like high-energy neutrons, there is little interaction with the IC package materials. Therefore, standard production packaging can be used for accelerated thermal neutron testing. The only exception is large metal heatsinks that might scatter the thermal neutron beam (JESD89A Sect. 7.7.5).

3.4.4 Thermal Neutron Sources: Selection, Calibration and Shielding Effects (JESD89A Sect. 7.4)

High flux thermal neutrons can be obtained from nuclear reactors and particle accelerators (JESD89A Sect. 7.4.1). An up to date listing of thermal neutron user facilities can be found at http://www.seutest.com/mwiki/index.php?title=Facilities. Calibration of the flux and energy measurements of thermal neutrons generated from an accelerator are critical because SER cross-sections can vary significantly with small changes in neutron energy. Since low energy neutrons are easily scattered, the concept of a well defined beam is not applicable – so the term "beam" is used loosely. Flux calibration of thermal neutrons from a reactor might already exist, but the impact of background gamma radiation on the device should be taken into consideration. ASTM E262-03 Standard Method for Determining Thermal Neutron Reaction and Fluence Rates by Radioactivation Techniques [9] addresses the issue of thermal neutron flux measurement.

Thermal neutrons can be shielded by using materials with high thermal neutron cross-sections – $_{10}$B, $_{113}$Cd and $_{157}$Gd. Table 7.1 of JESD89A gives examples of thermal neutron attenuation for these three materials. Attenuation of orders of magnitude can be obtained with films as thin as 1 mm.

3.4.5 SEU Cross-Section and SEU Rate (JESD89A Sect. 7.6.2)

The thermal neutron soft error averaged cross-section (σ_{nth}) of a device is defined as

$$\sigma_{\text{nth}} = \frac{N_{\text{nth}}}{\phi_{\text{acc}} T} \tag{3.12}$$

where N_{nth} is the number of errors induced by the accelerated thermal neutron source, ϕ_{acc} is the thermal neutron flux of the source in units of neutrons cm^{-2} s^{-1} averaged over the energy range of 0–20 eV and T is the test time duration (s^{-1}). It is assumed that the spectral distribution of the source is the same as the terrestrial thermal spectrum shown in Fig. 3.5. This is a difficult assumption to validate because the shape of the lower energy spectrum depends on how local materials scatter neutrons. Relative flux at thermal energies (<0.4 eV) does not correlate well with relative fluxes at higher energies as discussed in Sect. 3.4.1.

The thermal neutron soft error rate is given in JESD89A Sect. 7.6.2 as

$$R_{nhe} = \phi_{nth}\sigma_{nth} \tag{3.13}$$

where $\phi_{nth} = 1.8 \times 10^{-3}$ cm^{-2} s^{-1} is defined as the integrated reference level thermal neutron flux. This rate can be adjusted for specific locations and environmental conditions discussed in JESD89 Sect. A.4. The flux of thermal neutrons can be estimated within a factor of 2.3 by using this integrated reference value and scaling it with the high-energy neutron flux given by (3.2). But the actual thermal neutron flux in the field might differ from this estimation.

3.4.6 *Limitations and Strengths of the Accelerated Thermal Neutron Test*

The strong point of accelerated thermal neutron testing is that it allows for rapid determination if thermal neutrons play a role in the device soft error rate. If this is the case, special techniques can be used to mitigate the problem (e.g. elimination of BPSG films or isotopic replacement of $_{10}B$ with $_{11}B$).

Measurement of thermal neutron cross-section (3.11) can be used to determine a soft error rate referenced to a flux of 1.8×10^{-3} cm^{-2} s^{-1}. However, the difficulty is translating this to an actual field soft error rate at other locations. No quantitative expression is known for the thermal neutron flux as it is for high-energy neutrons (3.2) due to the interaction of the high-energy spectra with surrounding materials. Measurement of the thermal neutron flux at the desired location is required to translate the measured cross-section to an actual field use situation.

3.5 Real-Time (or Unaccelerated) Soft Error Testing

3.5.1 *Goal of the Test Method*

The term System Soft Error Rate (SSER) has also been used for real-time SER testing. The intent of the test is to characterize the error rate of a device in its real environment. This means that neutrons and alpha particles are present in quantities comparable to what the device would see in actual operating conditions. The total measure soft error rate (R_{total}) is given by

$$R_{\text{total}} = R_{\alpha} + R_{\text{nhe}} + R_{\text{nth}} \tag{3.14}$$

The alpha particle contribution is independent of environment and solely dependent upon packaging materials and manufacturing processes. The high energy and thermal neutron contributions are a function of the location and environment.

3.5.2 Large Sample Size and Long Test Period Required

Since the soft error rates can be quite low (100–10,000 FIT per chip for example), measurement of a large number of devices over a long period of time are required. A device showing a 1,000 FIT would have an error rate of 10^3 errors/10^9 h $= 10^{-6}$ error/h. It would take 1,000 devices running for 1,000 h to see just one error. So the drawback of Real-Time Soft Error Testing becomes immediately obvious for devices that show moderate to low soft error rates.

3.5.3 Separating the Alpha and Neutron Contributions to SER

Since alpha particle flux is independent of environment, underground testing can be used to shield the devices from the cosmic-ray neutron background. In this case, (3.14) simplifies to

$$R_{\text{total}} = R_{\alpha} \tag{3.15}$$

So underground testing allows for the determination of the alpha contribution to SER independent of neutron effects. The effects of thermal neutrons can be eliminated by shielding the device with $_{10}$B, $_{113}$Cd or $_{157}$Gd. Equation (3.14) then becomes

$$R_{\text{total}} = R_{\alpha} + R_{\text{nhe}} \tag{3.16}$$

Table 7.1 of JESD89A provides thermal neutron attenuation for several film thicknesses. Once R_{α} and R_{nhe} are known, the thermal neutron contribution can be determined by removing the shield material and measuring the total soft error rate (3.14).

3.5.4 Altitude Testing to Increase Neutron SER Contribution

Since the high-energy neutron flux is dependent on altitude (JESD89A Annex A), real-time SER testing can be done at high altitudes to increase error counting. Thermal neutrons flux will also scale with the high-energy flux, but building effects will add uncertainty to thermal neutron flux. Nevertheless, high-altitude testing has proved to be useful to many IC vendors in speeding up test times and reducing device count.

3.5.5 Building Shielding Effects (JESD89 Annex A.5)

Sections 3.1 and 3.2 cover the neutron spectra outdoors. However, with the possible exception of handheld mobile devices, almost all commercial IC applications reside inside buildings and equipment chassis, which can partially shield neutrons. Low-energy neutrons can be scattered and thermalized by these intervening materials, primarily concrete and steel. High-energy neutrons can be attenuated by interactions with the nuclei in the shielding material, causing emission of neutrons at lower energies. JESD89A Annex A.5 states that a 30 cm slab of concrete will attenuate the high-energy (>10 MeV) portion of the spectra by a factor of 2.3, but will only attenuate the total spectra by 1.6. However, recent studies [10] indicate that neutrons in the 1–10 MeV range can be attenuated by a CPU/heat-sink/motherboard stack by as much as 50–60%, whereas higher energy neutrons (100 MeV–1 GeV) are only attenuated by 20%. Part of this discrepancy might be due to the techniques used to measure the transmitted neutrons. If scattering takes place, the integration angle of neutron detector is important in measuring the total flux. Another factor could be the high Z materials (e.g. Cu and Pb) present in the stack, causing distortion of the neutron spectra.

Building materials such as concrete floors will tend to scatter low energy neutrons and attenuate high-energy (>10 MeV) neutron flux. An estimate of attenuated high-energy flux is

$$\phi = \phi_0 \mathrm{e}^{-2.7x} \tag{3.17}$$

where ϕ_0 is the unattenuated outdoor flux and x is the concrete thickness in meters. For a concrete density of 2.3 g cm^{-3}, the mass attenuation length of a concrete floor is 85 g cm^{-2}.

Large amounts of steel or other materials with high atomic number can distort the neutron spectra. Incident high-energy neutrons can liberate several lower energy neutrons, causing an actual increase in the low energy flux. Steel also absorbs thermal neutron. So a room with a significant content of steel in the floor and walls with have a significantly reduced thermal neutron flux. Several radiation transport codes exist to more accurately model the effects of thick concrete walls and other building materials (see Sect. 3.3.4).

3.5.6 Minimum FIT and Confidence Levels (JESD89A Annex C)

It is expected that the number of errors counted in a real-time SER test will be significantly less than any of the accelerated SER techniques. So quantifying the statistical uncertainty of the test is important. Annex C of JESD89 gives guidelines assuming a Poisson distribution

$$f(t) = N\lambda \mathrm{e}^{-N\lambda t} \tag{3.18}$$

Table 3.2 Example of estimate of soft error FIT at a 10% lower limit and 90% upper limit

T (h)	Number of devices	Errors (r)	R_{avg} (FIT)	R_{lower} (FIT)	R_{upper} (FIT)
1,000	1,000	1	1,000	355	4,744
2,000	1,000	2	1,000	409	3,148
3,000	1,000	3	1,000	455	2,585
4,000	1,000	4	1,000	493	2,288
5,000	1,000	5	1,000	523	2,103
6,000	1,000	6	1,000	548	1,974
7,000	1,000	7	1,000	569	1,878
8,000	1,000	8	1,000	587	1,804
9,000	1,000	9	1,000	603	1,745
10,000	1,000	10	1,000	617	1,696

where λ is the mean error rate, N is the number of devices in test and t is the test time. The upper and lower limit of the soft error rate expressed in FIT is given by

$$R_{\text{lower}} = \frac{X^2_{1-(\alpha/2);2(r+1)}}{2NT} \tag{3.19}$$

$$R_{\text{upper}} = \frac{X^2_{\alpha/2;2(r+1)}}{2NT} \tag{3.20}$$

where X^2 is the chisquare distribution, α is the confidence interval and r is the total number of errors counted in the time T. Table 3.2 shows a hypothetical result of a real-time soft error test using 1,000 components tested over a 10,000 h period for a 10% lower and 90% upper confidence limit (i.e. there is only a 10% probability the true error rate is below R_{lower} and a 90% probability that it is below R_{upper}). With an average error rate of 1 per 1,000 h from 1,000 devices, the table shows a simplified situation, where an error occurs every 100 h. A real error count would not be so predictable, but the purpose of Table 3.2 is to show how the upper and lower estimates of the soft error rate converge to the mean as more errors are accumulated.

3.5.7 Limitations and Strengths of the Real-Time Test

The primary strength of real-time SER testing is that it bears a well-defined correlation with real life use of the devices, since all three radiation sources (alpha, high energy and thermal neutrons) are present. However, translation of the soft error rates from one geographical location to another is not possible unless the contribution of each radiation source is known (Sect. 3.5.3). This requires accelerated alpha and neutron testing (or real-time testing in an underground

location with no neutron flux in addition to two different locations with different high energy and thermal neutron fluxes).

The primary drawback of real-time SER test is the amount of time and number of devices required to obtain an accurate estimate of the soft error rate. Devices that have low soft error rates require a larger number of devices over a longer period of time. Since the results of a single test provide no insight as to the primary source of soft errors – alpha particles, high-energy or thermal neutrons, it would be difficult to determine the best approach needed to reduce the soft error rate of a device if the specifications require it.

3.6 Conclusions

The purpose of the JESD89 series standards is to provide comprehensive procedures to characterize the soft error rate of ICs in terrestrial environments. The sources of these errors include radiation from alpha particles, high-energy neutrons and thermal neutrons. JESD89A is the over-arching standard that covers all three causes of soft errors and their origins in detail. JESD89-1A, JESD89-2A and JESD89-3A target-specific soft error test techniques. Their intent is to provide simplified test techniques that allow IC vendors (or IC consumers) with limited resources the ability to partially characterize a devices soft error rate. In order to characterize the complete soft error rate of an IC, the effects of alpha particles, high-energy neutrons and thermal neutrons must be measured. A real-time soft error test (JESD89A Sect. 4 and JESD89-1A) is sufficient to capture all three sources assuming enough device hours can be accumulated to render a statistically meaningful measurement. However, these results alone do not allow for the translation of the soft error rate to different locations and environments. An independent measurement of each of the individual components of the soft error rate is required to do this. Alpha particle SER is independent of location. High-energy neutron SER scales with the location and JESD89A provides methods for this scaling (JESD89A Annex A). Thermal neutron SER will also scale with location, but there is an added uncertainty in flux due to environmental effects on generation and absorption (see JESD89A Fig. A.4.1).

These standards are relatively new (the first revision of JESD89 was released by JEDEC in August 2001) even though the history of soft errors in commercial semiconductor devices spans over three decades. The standards have been leveraged off of considerable knowledge gained in the aerospace community and semiconductor industry over the past several decades. The JESD89 standards are now widely referenced in technical publications on soft error rates in commercial ICs. It is safe to say that these standards will continue to undergo further revisions as the knowledge base of the IC industry increases and devices scale into the sub-100 nm region.

References

1. www.jedec.org/join_jedec/history.cfm.
2. JEDEC Online Resource Guide at www.jedec.org.
3. T.C May and M.H. Woods, "A new physical mechanism for soft-errors in dynamic memories", Proc.16th Annual Reliability Physics Symp, Apr. 1978, pp. 33–402.
4. T.C. May and M.H. Woods, "Alpha-particle-induced soft error in dynamic memories", IEEE Trans. Electron Devices, vol. 26, no. 1, pp. 2–9, Jan. 1979.
5. C.S. Guenzer, E.A. Wolicki, and R.G. Allas, "Single event upset of dynamic RAMs by neutrons, protons", IEEE Trans. Nucl. Sci., vol. 26, pp. 5048–5052, Dec. 1979.
6. J.F. Ziegler, "SRIM 2003," Nucl. Instrum. Methods, vol. 219–220, pp. 1027–1036, 2004.
7. K.M. Warren, J.D. Wilkinson, R.A. Weller, B.D. Sierawski, R.A. Reed, M.E. Porter, M.H. Mendenhall, R.D. Schrimpf, and L.W. Massengill, "Predicting neutron induced soft error rates: Evaluation of accelerated ground based test methods", IEEE 46th Annual International Reliability Physics Symposium, Phoenix AZ, May 2008, pp. 473–477.
8. M.S. Gordon, P. Goldhagen, K.P. Rodbell, T.H. Zabel, H.H.K. Tang, J.M. Clem, and P. Bailey, "Measurement of the flux and energy spectrum of cosmic-ray induced neutrons on the ground", IEEE Trans Nucl. Sci., vol. 51, no. 6, pp. 3427–3434, Dec. 2004.
9. ASTM E262-03 Standard method for determining thermal neutron reaction and fluence rates by radioactivation techniques.
10. A. Dixit, R. Heald, and A. Wood, "Trends from ten years of soft error experimentation", IEEE Workshop on Silicon Errors in Logic – System Effects, Stanford Univ, CA, March 24–25, 2009. Available online at www.selse.org/Papers/selse5_submission_29.pdf.

Chapter 4
Gate Level Modeling and Simulation

Nadine Buard and Lorena Anghel

This chapter presents an overview of a methodology for analyzing the behavior of combinational and sequential cells regarding "Single-Event Multiple-Transients" (SEMT) caused by nuclear reactions induced by atmospheric neutrons. The methodology uses a combination of Monte Carlo-based selection of nuclear reactions, simulation of the carriers transport in the device, and SPICE simulation. The effects of nuclear particles on the gates are monitored at the gate output by means of transient duration, amplitude, and associated occurrence probability.

4.1 Introduction

Heavy ions and protons interactions have been considered in the past as one of the most important challenges in space electronics as they induced an increased number of application failures. In nowadays race for smaller device and wire silicon technologies, transients induced by neutrons interactions with silicon are a serious concern, not only for space and avionic applications [1, 2], but also for ground level commercial ones [3, 4]. Transients may flip the content of latches, flip-flops and SRAM memory cells (we call this phenomenon SEU or Single Event Upset), or they may provoke Single Event Transient (SET) or Single-Event Multiple-Transients (SEMT) within combinational cells, which can be propagated through the combinational logic and be captured by sequential cells (i.e., latches or flip-flops). Because of the statistical nature of the nuclear interactions, electrical attenuation, and logic and architectural masking, the computation of Soft Error Rate (SER) is still a challenge, especially in complex ICs. Early estimation of SER is mandatory in order to propose efficient resilience techniques to protect vulnerable designs.

N. Buard (✉)
Head of Electronic Systems Department, Head of Operations Sensors, Electronics and System Integration (TCC4) EADS Innovation Works, 12 rue Pasteur, 92152 Suresnes, France
e-mail: nadine.buard@eads.net

M. Nicolaidis (ed.), *Soft Errors in Modern Electronic Systems*,
Frontiers in Electronic Testing 41, DOI 10.1007/978-1-4419-6993-4_4,
© Springer Science+Business Media, LLC 2011

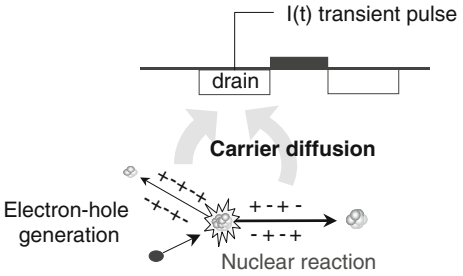


Fig. 4.1 Physical mechanisms responsible of transient pulses

An accurate system SER modeling of a chip or its components require an in-depth knowledge of the constituent latches, memory arrays, and combinational logic with respect to their associated sensitivity to soft errors. On the basis of the SER computation are the transient pulse generation and propagation within a gate, which can be accurately analyzed by combined Monte Carlo device simulations and SPICE simulations.

Neutrons are well identified to be responsible of single events in many types of electronic components. Though neutrons are not able to induce direct ionization, their nuclear reactions with the die materials produce secondary particles (Fig. 4.1), which induce charge deposition and subsequently charge collection and transient currents.

Another source of single events is the alpha particles emitted by radioactive impurities contained in packages and materials of the chip.

To be able to predict the sensitivity of memory arrays, sequential and logic gates in a specific technology, different steps have to be taken into account (Figs. 4.2 and 4.3).

1. *Nuclear interactions, perturbations-induced and associated probabilities*:
   Several methods were developed to compute the error rate induced by neutron or proton environment in CMOS logic devices, either analytical or semi analytical, or based on Monte Carlo selection of nuclear reactions. The computation code consists in coupling the nuclear reaction databases (neutron or proton with silicon) with a technological description, and then to calculate the transient current pulse produced in the drain of a MOS transistor thanks to a diffusion–collection model. Monte Carlo selection applied to nuclear reaction location and types allows obtaining current pulses library associated to a given transistor and its probabilities.
2. *Selection of the transient currents able to disturb the output of the gate*:
   The next step is to select only the statistically relevant transient currents by using some criteria. SPICE simulations at transistor level are used to inject each current pulse from the current pulse library in the different sensitive zones of the design and obtain a realistic transient behavior in terms of amplitude and duration at the output of circuit with corresponding probabilities.
   Each step of the simulation and the corresponding results is described in details in the following paragraphs.

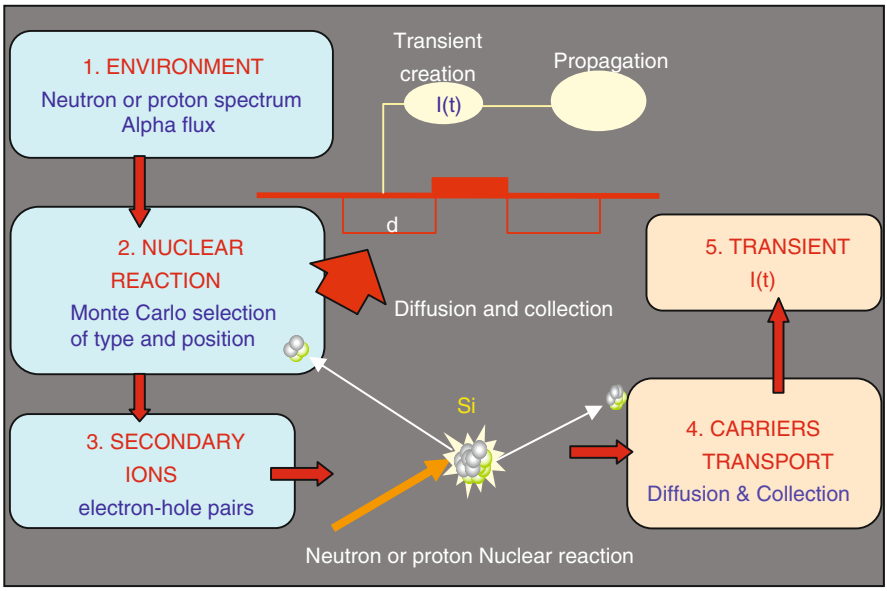


Fig. 4.2 General chart of the different steps required to compute the library of transient currents for each transistor

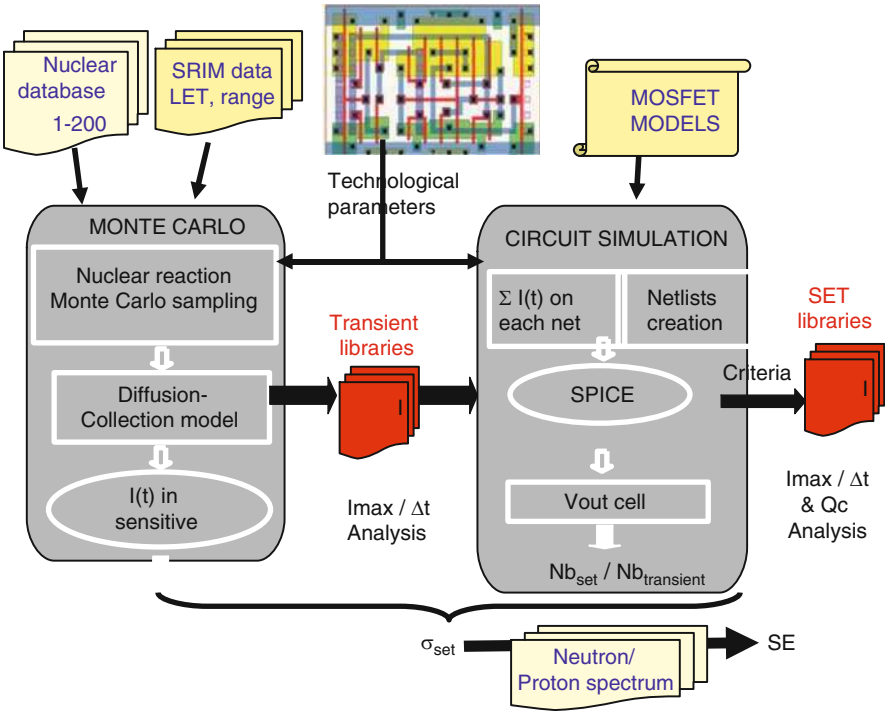


Fig. 4.3 Coupling of Monte Carlo and SPICE simulation to compute SET and SEMT rates in logic gates

4.2 From Nuclear Interactions to Transient Current Computation by Monte Carlo Selection of Nuclear Reactions and Device Simulation

As described in the Introduction, neutrons in the atmosphere have a very wide spectrum in energy, from MeV to GeV, and flux is isotropic in direction. Any of these neutrons coming from a given direction is able to interact with any of the atoms from the electronic devices, with different nuclear reactions possible, each of them producing several types of ionizing ions in several directions.

Given the statistical properties of the phenomena, we need to compute the probability to have a given type of event. Thus, Monte Carlo simulation is necessary to compute SER (Fig. 4.4).

One possibility is to use nuclear interaction code like GEANT4, able to follow the history of one particular particle among its interactions with matter, then, compute the impact of such interactions on the cell under investigation.

Simpler methods have been proposed, combining nuclear codes and device simulations or semiempirical coupling of nuclear physics and experimental data [5–7], especially the famous BGR method proposed by Ziegler, Lanford, and Normand. In 1998, Vial et al. [8] built a neutron–silicon nuclear database and in

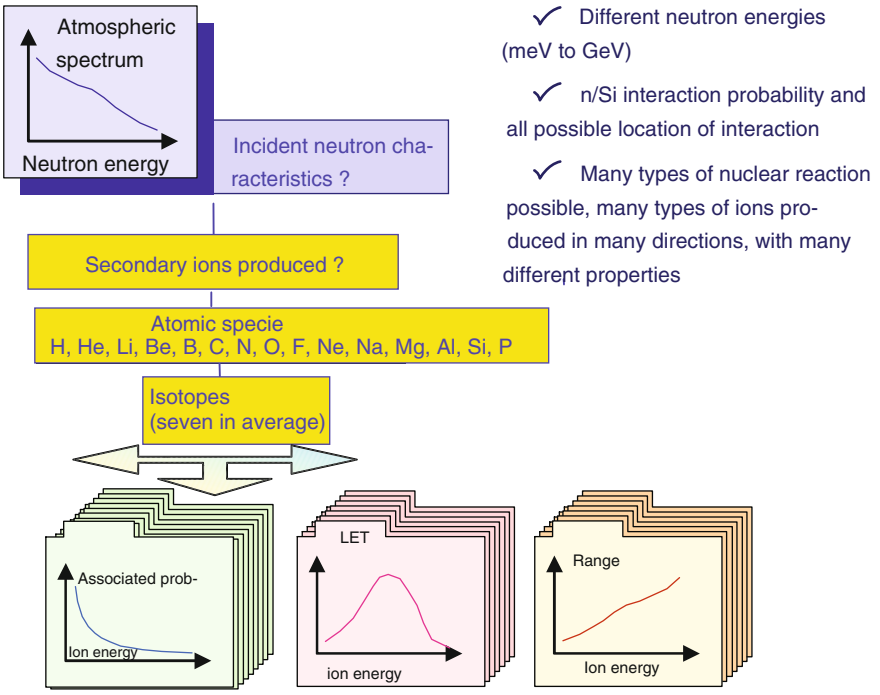

Atmospheric spectrum
Neutron energy
Incident neutron characteristics ?
✓ Different neutron energies (meV to GeV)
✓ n/Si interaction probability and all possible location of interaction
✓ Many types of nuclear reaction possible, many types of ions produced in many directions, with many different properties
Secondary ions produced ?
Atomic specie
H, He, Li, Be, B, C, N, O, F, Ne, Na, Mg, Al, Si, P
Isotopes
(seven in average)
Associated prob-
Ion energy
LET
ion energy
Range
Ion energy

Fig. 4.4 Schematic of the possible secondary ions created by atmospheric neutron spectrum in Silicon

2001, Hubert et al. [9] proposed a new approach based on nuclear database coupling with occurrence criteria deduced from a Detailed Analysis of Secondary Ion Effect (DASIE). This method was first dedicated to SEU investigations in SRAM memory cells. Since 2001, different DASIE methods were derived, especially one method allowing Multiple Cell Upset (MCU) prediction, called SMC DASIE [10, 11], dedicated to SRAM device simulation. Similar methods have been developed by Ibe et al. from Hitachi [12] (SECIS, Self-consistent Integrated system for terrestrial neutron soft error) and Murley et al. from IBM [13, 14] (SEMM, Soft-error Monte Carlo modeling).

But the most efficient way to simulate such physics was shown to be Monte Carlo selection of possible nuclear reactions in a nuclear database performed, thanks to nuclear physics code. This selection, done only once, allows the reduction computation time by several orders of magnitude. The last version of IBM software, called SEMM2 is based on this principle, as well as the Transient Monte Carlo Detailed Analysis of Secondary Ions Effects (TMC DASIE) software from EADS whose results are used below for demonstration.

4.2.1 Database of Neutrons/Matter Nuclear Interactions

Nuclear reactions between neutron (or proton) and silicon nuclei can appear anywhere in the volume. Thus, the position of the nuclear reaction has to be Monte Carlo selected. A nuclear reaction is characterized by a probability which depends on the nucleon energy.

For example, nuclear databases of neutron–silicon interactions were built by using a Monte Carlo code, called MC-RED1.2 developed by F. Wrobel [15], for neutrons in the 1–200 MeV energy range. This code treats nonelastic reactions with the exciton model (pre-equilibrium step) and the Hauser–Feschbach formulation (equilibrium step). Elastic reactions are treated with the ECIS code by inputting an optical model from Recommended Input Parameters Library (RIPL). Nuclear databases comprise:

- The detail of the nuclear reactions, that is, the atomic and mass number of the secondary ions, their energies and directions
- The nuclear cross-section values relative to the elastic (σ_{elastic}) and nonelastic ($\sigma_{\text{nonelastic}}$) reaction types

In silicon, secondary particles created by nuclear interactions can range from H to P, including several isotopes, and several secondary ions are simultaneously generated by a specific nuclear reaction. The number of secondary ions produced depends (but not only) on the nucleon energy, for example, a 63 MeV neutron can create 1–4 secondary ions when 14 MeV neutron creates only 1–2.

Depending on its energy and atomic and mass number, each of these ions has a specific ionizing power in silicon, called Linear Energy Transfer (LET). Those values are tabulated in “Ziegler” tables and can be accessed, thanks to the SRIM

	A	Z	Energy	Angle		
Reaction n.1	1	1	35.716197	0.5553239	-0.19152668	0.80927925
	27	13	1.0527068	-0.72592564	0.21600247	0.65297389
Reaction n.2	28	14	0.32140786	0.16505668	0.86145195	0.48026745
Reaction n.3	1	1	4.6670238	-0.78903027	-0.31864736	0.52525717
	1	1	3.1638424	0.60223712	-0.3086832	0.73622356
	26	12	7.5125566e-002	0.75397569	-0.28900143	0.58991427
Reaction n.4	1	1	2.1810132	0.64681762	0.69866609	-0.30576568
	27	13	1.6931279	-0.11748162	0.55400652	0.82418132
Reaction n.5	1	1	7.1216641	0.2470587	0.28025365	-0.92758821
	26	13	5.8148175	-1.3500547e-002	-5.6288002e-002	0.99832329
Reaction n.6	1	1	38.842124	0.19809158	-0.34338335	0.91806732
	28	13	0.29360033	-0.43679733	0.75716963	0.48569769
Reaction n.7	4	2	13.721004	0.46724656	-0.19308608	0.86278527
	24	12	0.75840499	-0.561092	0.43116927	-0.70658958
.						
.						
.						
Reaction n.1E5	4	2	9.6824548	0.58840911	-6.8708606e-003	0.80853418
	23	11	0.78754302	-0.75813812	-0.62047341	-0.20059746
	1	1	22.697227	0.20776	0.15732681	0.96544501

Fig. 4.5 Example of the nuclear database, reactions induced by neutrons of a given energy

software. Large variations in the secondary ions properties generated by nuclear reactions are observed, but one has to consider both heavy and light ions. For example, for the same energy, a heavy secondary ion, which has a high LET and a short range, is more likely to cause a transient pulse than a light one, whose LET is smaller and range greater. But to observe a single event effect, the heavy ion requires to be generated close to the sensitive region while the light ion can be generated even farther with respect to the sensitive region. Silicon recoils provoke transient close to their generation point, while light ions, particularly He and H, can be created several tens or hundreds of micrometers away from the collection point (Fig. 4.5).

Nuclear interactions database can be generated with other elements compared to silicon. Indeed, other types of atoms than silicon are present inside electronic components. One of the major ones is not only Oxygen (SiO_2 for dielectrics and STI), but also Tungsten for plots, Copper for contacts. . . The relative importance of the different constituents was investigated by different authors, and depends on the technology of the chip. Nuclear reactions with oxygen atoms are usually not considered separately since they are very similar to the ones with silicon atoms.

4.2.2 Transient Currents Induced by Secondary Ions

Secondary ions created by nuclear reactions induce electron-hole pairs along their track by Coulombian interactions with electrons [16].

3D full-cell device simulations were performed with various track configurations [17, 18]. These 3D device simulations show that transient current pulses are due to the collection of charges by junctions. Those charges are the result of ambipolar diffusion of the original charges created by the heavy ion tracks.

When the ion track is sufficiently far from the space charge zone of the drain junction, the carriers generated in the tracks mainly move by diffusion. We consider that recombination is negligible and that the space charge zone of the drain collects all the carriers that cross it. Let us consider some simple geometry cases for which the pure diffusion equation can be solved analytically.

$$\frac{\partial n}{\partial t} = D\Delta n \tag{4.1}$$

where n is the density of electrons in the conduction band, obviously equal to the hole density p and D the ambipolar diffusion constant. The simple case is the case of spherical diffusion, i.e. electron-hole pairs deposited in a very small volume.

An ion track can be considered as a collection of a large number of punctual carrier densities n_D^i. The location of each punctual carrier density is characterized by a distance ρ_i from the drain. Each punctual carrier density is governed by a spherical diffusion law allowing the calculation of the carriers density arriving in the collection area. This carrier density is given by:

$$n(\mathrm{d}x, \mathrm{d}y, t) = \sum_i n_D^i \frac{\mathrm{e}^{-\frac{\rho_i^2}{4Dt}}}{(4\pi Dt)^{3/2}} \tag{4.2}$$

The current collected by the drain is calculated with the following formula:

$$I(t) = \int\int_{\text{drain}} qn(\mathrm{d}x,\ \mathrm{d}y,\ t)v\mathrm{d}x\mathrm{d}y \tag{4.3}$$

where q is the electrical charge (1.6E-19 C), dxdy the elementary surface, and v the collection velocity. Figure 4.6 presents the $I(t)$ transient calculation method.

The current pulse is delayed with respect to the impact time, and the delay depends on the distance between the track and collecting drain (Fig. 4.7). The delayed arrival of the current peak is basically due to the time needed for the excess carriers to reach the collecting areas. Obviously, there is no delay when a ion shunts the electrodes and the charge collection can begin immediately, resulting in a very short transient.

The drain currents induced by secondary ions and obtained by 3D device simulation were compared with a diffusion collection model based on ambipolar diffusion laws and good agreement was obtained [17]. As a conclusion, diffusion phenomenon was clearly identified as the main mechanism in transient pulse generation.

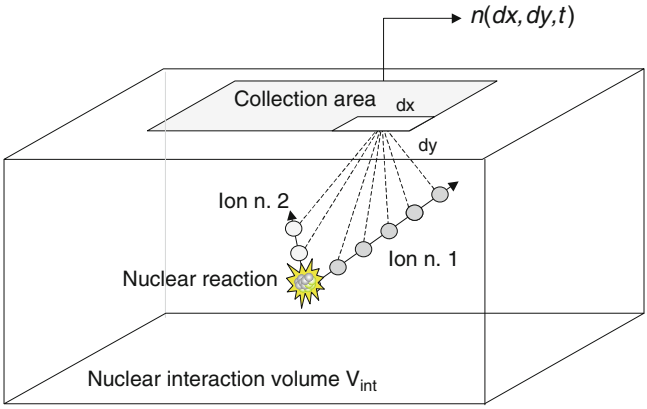


Fig. 4.6 Current pulse calculation method

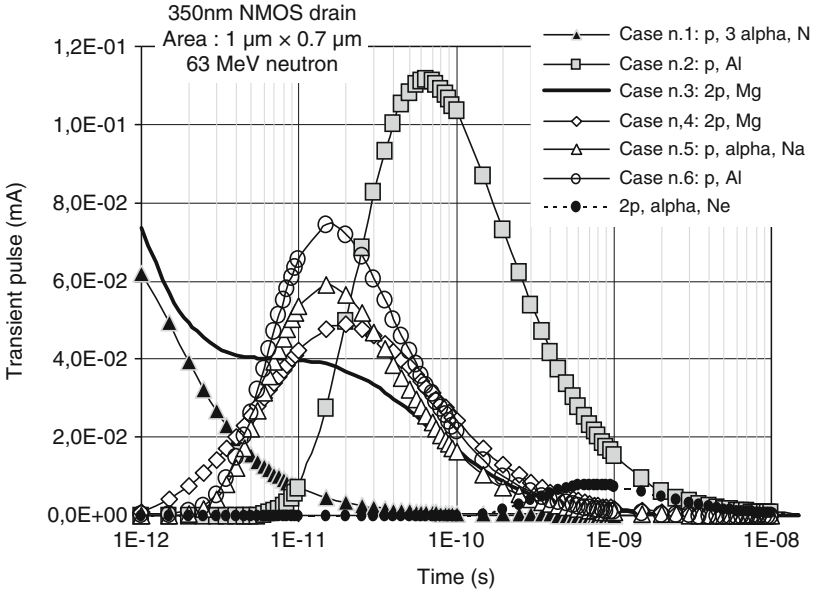


Fig. 4.7 Some currents (SET) obtained by TMC DASIE calculation applied to 350-nm technology and for 63-MeV neutron

4.2.3 Example: SEU and MCU Induced by High-Energy Neutrons in SRAM

As an illustration, these methods can be used to compute the upset rate of SRAM cells in order to explore the sensitivity trends of integrated technologies. In this case, the upset criteria is linked to the quantity of charges in the current pulse.

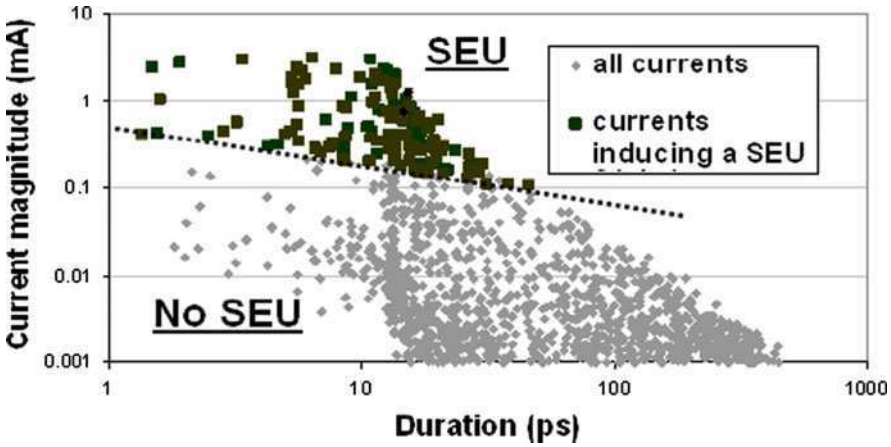


Fig. 4.8 Pulse magnitude versus duration. In *black*, currents which induce an upset. Results obtained on the NMOS sensitive zone of a 130-nm SRAM for 100-MeV neutrons

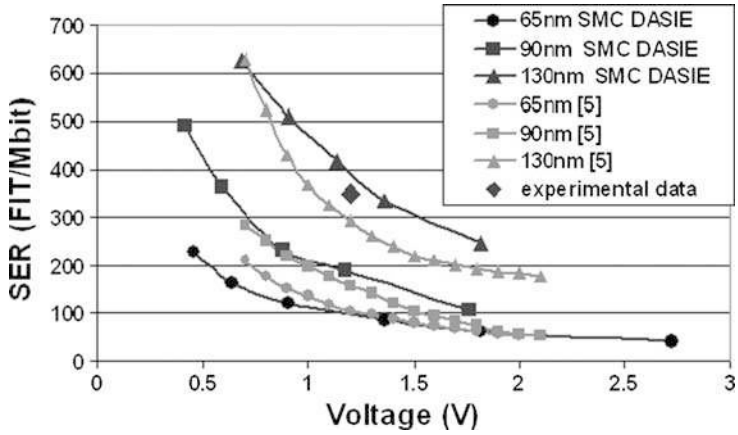


Fig. 4.9 Flips SER as a function of voltage induced in a SRAM by neutrons at ground level (in *gray*, normalized data from [19]; in *black*, SMC DASIE data)

The Fig. 4.8 presents the library of pulse magnitude versus the pulse duration for a given energy applied to the NMOS sensitive zone of a 130-nm SRAM. The currents which induce a SEU are represented in black.

In Fig. 4.9 simulation results on SRAM are compared with published simulation data [19] and proprietary experimental results. Input parameters (given in Table 4.1) vary according to process scaling.

Flips events are the total number of bit flips cumulated for SEU and MCU events.

4.3 Monte Carlo Simulation of SET and SEMT in Logic Gates

4.3.1 The Necessity to Take into Account Multiple Transient Currents Induced by a Single Event

In an ASIC library, physical designs vary drastically for every cell in terms of sensitive zone number, size, and geometry. As an example, an OAI22 (four input OR AND INVERTED) gate is made of 8 transistors and 13 sensitive zones.

For comparison, a nuclear reaction can induce alpha particles (atomic number $Z = 2$). If this particle has 2 MeV energy, it would have a range of 7.5 μm. Compared to OAI22 size, this particle would be able to induce charges simultaneously in all of the 13 sensitive zones (see Fig. 4.10). As technology is scaling down, sensitive zones become closer, and so cells are more vulnerable to SEMT.

To be able to characterize the sensitivity of complex cells, it is necessary to perform simulations for each gate at layout level, and to inject collected charges simultaneously in every sensitive zone of the gate.

The charges induced by the nuclear reaction and collected by the different cell sensitive zones can be computed by adequate Monte Carlo-based software that combines the nuclear databases with the layout view and some technological parameters.

Table 4.1 Reference values for technological parameters

Gate length (nm)	Critical charge (fC)	Critical LET (fC/μm)	Drain length (μm)
350	12	25	1
250	7	16	0.65
180	4.5	13	0.5
130	3	11	0.3
90	2	8	0.15

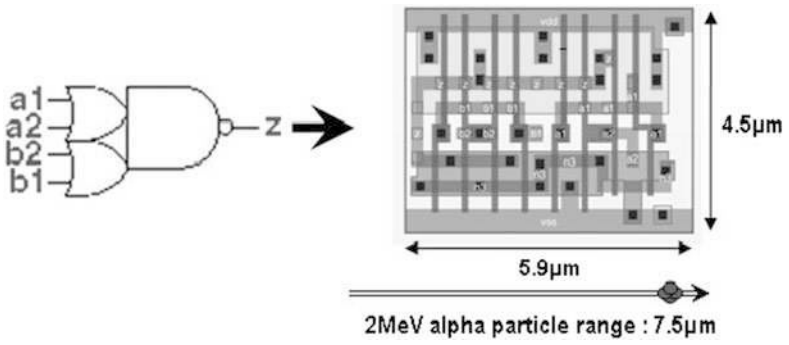


Fig. 4.10 Generic 130 nm four input OR AND INVERTED gate

4.3.2 Topological and Technological Description

The methodology used to extract sensitive zone geometries and organization is described below.

Starting from the layout view of the cell, every drain (or source) is located. As charges collected on zones tied to the power supply (V_{DD} or GND) are evacuated and will not provoke any effect, they are discarded from analysis. Figure 4.11 is an example of NAND2 gate in 130-nm technology. From the layout view, six zones are identified. Both PMOS sources are connected to the V_{DD} rail, thanks to tungsten plots, and the zone on the bottom left hand side is connected to the GND rail, with the same kind of plot. Thus, there are three remaining zones, which have a non-V_{DD} or non-GND voltage that can be potentially disturbed by the collection of additional charges.

The cell description obtained from layout view gives for each sensitive zone the transistor type (NMOS or PMOS), its dimensions (l_x, l_y), as well as the distance between the different sensitive zones. The depletion depth value, given by the manufacturer, is also required to model the volume.

Sensitive zones are surrounded by a semiconductor layer and an insulator layer, with almost the same depth. Sensitive zones are on the top of the semiconductor layer (Si), and the insulator layer (SiO_2) is above them (see Fig. 4.11). As particles (especially light ones) created by a nuclear reaction can travel through several micrometers, it is necessary to model an environment which is large enough: $12 \times 12 \times 12$ µm in the following examples. To model accurately actual ICs structures, a small insulator layer is considered between sensitive zones, which represents shallow trench insulators (STI).

Inside SiO_2, secondary ions can be created but their charges are trapped in the insulator, and so they do not diffuse. It is true for insulator layer of SOI, and for STI.

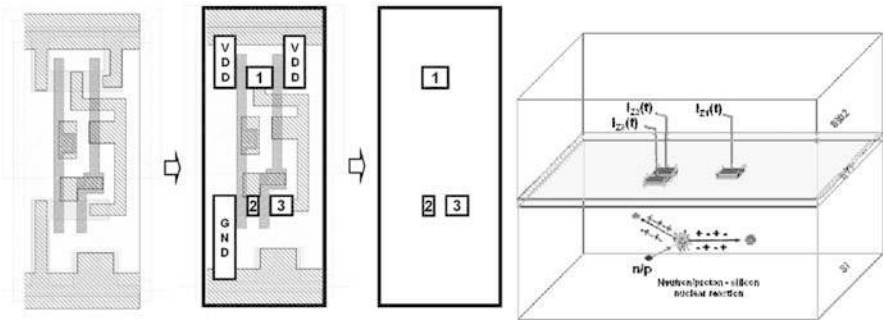


Fig. 4.11 Sensitive zone extraction of ATMEL 130-nm NAND2 gate and simulated environment around sensitive zones

4.3.3 Nuclear Reaction Example

The following example shows a nuclear reaction produced by a neutron with an initial energy of 63 MeV. The case is an inelastic reaction occurring in the SiO_2 layer. Three particles are created (see Fig. 4.12):

- One 0.3 MeV proton ($Z = 1$, range $= 2.9$ µm)
- One 4.8 MeV proton ($Z = 1$, range $= 200$ µm)
- One 2 MeV magnesium ion ($Z = 12$, range $= 2.2$ µm)

The recoil neutron is not taken into account because it has a very low probability to induce a second nuclear reaction.

The first proton goes in the insulator part. On the other hand, the two other particles cross the silicon through several micrometers and their carriers diffuse inside, following a spherical diffusion law.

Figure 4.13 shows the collected current pulses on each of the three zones of Fig. 4.12. As ionizing particle trajectories are closely located to zone 2, it is the first one to collect charges ($I2(t)$). Zone 3 is very close to the reaction and has a larger volume than zone 2, and so it also collects an important amount of charge, with a small delay ($I3(t)$).

Zone 1 is too far from particle trajectories, and so it collects less charges, which are delayed by several picoseconds after the others ($I1(t)$).

Since the zone 3 is larger than zone 2, it collects more charges during a longer time (51fC and 41fC, respectively), even if current amplitude is smaller. Zone 1 collects only 11fC.

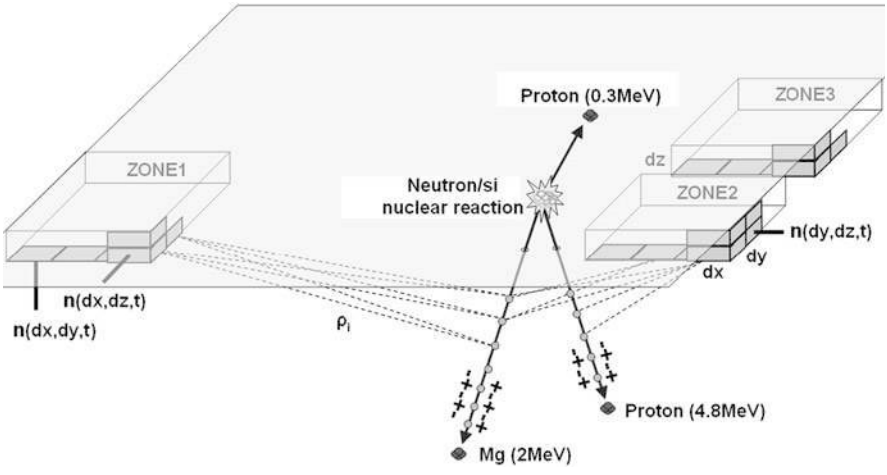


Fig. 4.12 Nuclear reaction example from 63-MeV neutron

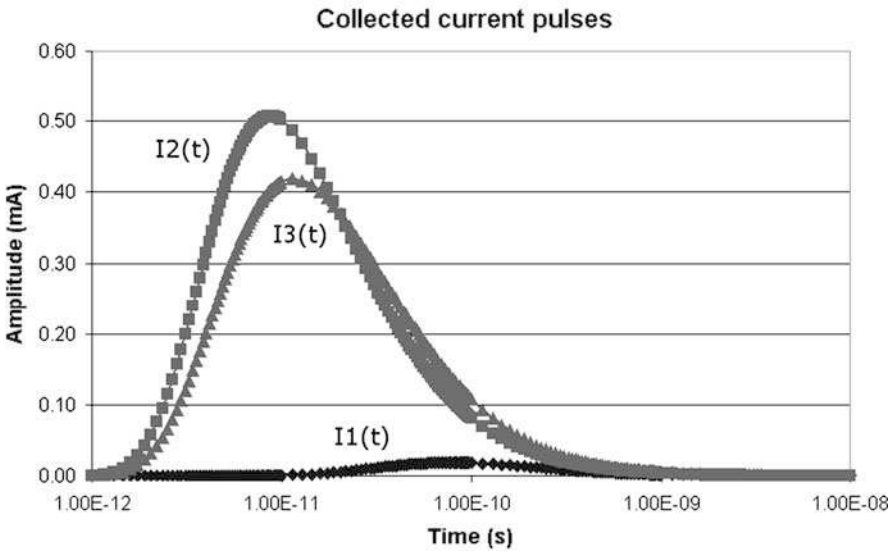


Fig. 4.13 Collected current pulses for each zone from 63-MeV neutron reaction

4.3.4 Transient Pulse Calculation

Transient pulse calculation is based on two different methods, depending on whether or not ion tracks cross-sensitive zones. On the first case, carriers are collected by drift phenomenon, and we consider that the space charge areas of the sensitive zones collect all the carriers that cross it. On the other case, carriers diffuse in the semiconductor material, and we use the spherical diffusion law [9], considering that recombination phenomenon is negligible.

Ion tracks are modeled as a multitude of punctual carrier densities n_D^i. The location of each punctual carrier density is characterized by a distance ρ_i from the collected area. Each punctual carrier density is governed by a spherical diffusion law allowing the calculation of the carrier density arriving in the collected area. As discussed in Sect. 4.2.2, this carrier density is given by (4.4):

$$n(\mathrm{d}x, \mathrm{d}y, t) = \sum_i \left(n_D^i \frac{\mathrm{e}^{-\frac{\rho_i^2}{4Dt}}}{(4\pi Dt)^{3/2}} \right) \tag{4.4}$$

where D is the diffusion coefficient (20 cm^2/s).

Current collected for the different zones are calculated with these following formulas (4.5):

$$\begin{aligned} I1(t) &= qv\left(\iint_{x1,\,y1} n(\mathrm{d}x,\, \mathrm{d}y,\, t)\mathrm{d}x\mathrm{d}y + \ldots + \iint_{y1,\,z1} n(\mathrm{d}y,\, \mathrm{d}z,\, t)\mathrm{d}y\mathrm{d}z \right) \\ I2(t) &= qv\left(\iint_{x2,\,y2} n(\mathrm{d}x,\, \mathrm{d}y,\, t)\mathrm{d}x\mathrm{d}y + \ldots + \iint_{y2,\,z2} n(dy,\, dz,\, t)\mathrm{d}y\mathrm{d}z \right) \\ I3(t) &= qv\left(\iint_{x3,\,y3} n(\mathrm{d}x,\, \mathrm{d}y,\, t)\mathrm{d}x\mathrm{d}y + \ldots + \iint_{y3,\,z3} n(\mathrm{d}y,\, \mathrm{d}z,\, t)\mathrm{d}y\mathrm{d}z \right) \end{aligned} \tag{4.5}$$

where q is the electrical charge (1.6E-19 C), dx, dy, and dz the elementary surfaces of each sensitive zone sides, and v the collection velocity.

Only collected currents with at least 10 μA of amplitude are recorded. For each zone, a library of several thousand current pulses, with the associate occurrence probability can be generated.

Each transient is associated with an occurrence probability depending on the nuclear reaction type and calculated by (4.6),

$$\sigma_{\text{transient}} = \sigma_{\text{nucl}} N_{\text{at}} V_{\text{int}} \frac{nb_{\text{transient}}}{nb_{\text{nucl react}}} \tag{4.6}$$

with N_{at} the atomic density in atoms/cm^3, V_{int} the interaction volume in cm^3, σ_{nucl} the elastic or nonelastic nuclear cross-section in cm^2, $nb_{\text{transient}}$ the number of transients observed, and $nb_{\text{nulc.react}}$ the total number of nuclear reactions.

4.3.5 Current Pulse Statistics

The largest sensitive volumes are able to collect the maximum amount of charges. Indeed, if the same calculation is done with the whole library, it shows that the largest sensitive volume (zone 1) is most often the one which collect the maximum charge. For the three sensitive zones of the NAND2 gate, Table 4.2 presents the collected charge and the percentage of the associated volume to each sensitive zone.

The frequency distribution of collected charge on each zone shows that most of them are below 3fC (Fig. 4.14). This histogram also shows that the maximum amount of charge collected by the sensitive zones depends on their volume.

4.4 SPICE Analysis Methodology to Estimate SER Probability for Sequential and Combinational Cells

The current pulse library generated for each specific gate is used during SPICE simulation in order to inject the current pulses and determine the duration of the corresponding voltage pulses produced at the output of the gate together with their occurrence probabilities.

The simulation methodology is presented below (Fig. 4.15).

Table 4.2 Maximum charge collected on a 130-nm NAND2 gate

Zone #	Maximum charge collected (%)	Volume (%)
Zone 1	46	43
Zone 2	23	21
Zone 3	31	36

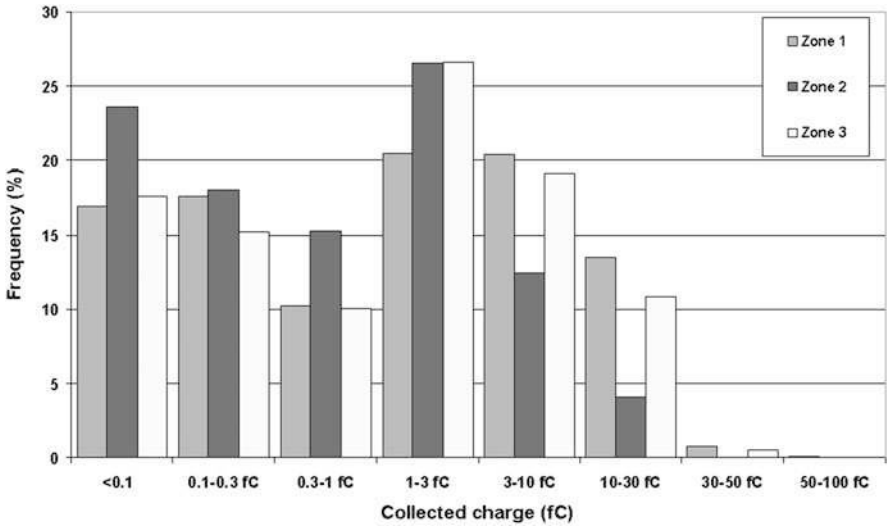


Fig. 4.14 Collected charge histogram obtained on the 130-nm NAND2 gate

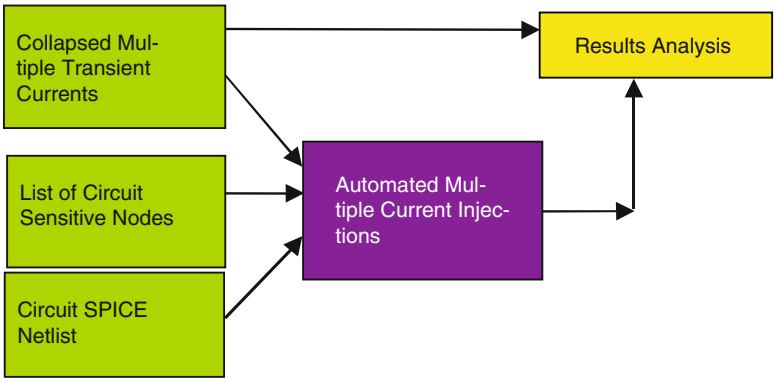


Fig. 4.15 Multiple transient currents simulation

For this analysis, (1) the cell SPICE netlist, (2) the sensitive zone list or the injection list, and (3) optimized (collapsed) libraries of the transient pulses, have to be considered.

By taking into account circuit layout characteristics, the different sensitive zones are identified by TMC DASIE v1.1, and libraries of several thousand transient pulses are generated. For a given design, transient currents may vary drastically in terms of shape, duration, and amplitude. To obtain a faster evaluation, the total number of currents considered for the injection campaign has to be reduced. By a careful analysis of currents parameters and characteristics (shape, amplitude, and duration), currents are classified in several classes and a Collapsed Current Library is obtained.

The *Automated Multiple Current Injections* module implements several modifications of the original SPICE netlist of the circuit consisting in automatically adding current sources for the identified sensitive zones. In order to be closer to the reality, the SPICE piecewise linear waveform (PWL) current format is used to allow the injection of more realistic currents. The reason we use current injection simulation by means of PWL files is not only the automation process and easiness of fault injection simulation, but also computation accuracy. Indeed, so far, researchers have considered that transient currents obey double exponential formula [20], which is not always the case for currents induced by the secondary ions of neutrons and protons reactions.

The *Result Analysis* module processes and analyses the simulation results. In the following, we detail each part of the analysis methodology.

4.4.1 Collapsed Transient Currents Analysis

Analyzing and collapsing multiple transient currents is very important, especially when the list of transient currents generated by TMC DASIE has an important size, which is generally the case for neutrons and protons interactions. Collapsing allows drastic reduction of the simulation time, as it reduces drastically the number of transient currents to inject and simulate. For performing transient currents classification, several steps have to be considered: because we have to consider multiple current pulses induced by a single event, we will have to treat *n*-tuples of current pulses.

Before creating classes for the *n*-tuples of currents, we first classify the individual current pulses. The first classification is based on the shape of the transient currents. Examples of current shapes can be seen in Fig. 4.7. Several types of current shapes are first defined. Then, for each type of shape, we consider different characteristics of the transient current (amplitude, slope, and duration at 30% of the amplitude). Using these criteria, we classify the pulses of a given type by specifying the intervals of amplitude, slope, and duration of each class.

This step allows a fast and flexible customization of the equivalence classes. Each class of transient currents will be represented by a single transient current. Thus, we can reduce drastically the number of currents to be simulated. The user will select the number of equivalence classes and will also select for each class the three parameters of the equivalence criterion (by considering simulation time versus accuracy tradeoffs). If the user selects a large number of classes and attributes narrow intervals of values to each of the parameters defining a class, a higher accuracy will be obtained at the cost of a higher simulation time. If a small number of classes is selected and large intervals of values are set for each of the parameters defining a class, the simulation time is reduced drastically at the cost of lower accuracy. Usually, a middle solution is the most suitable.

Once the currents are individually classified, we further classify the current *n*-tuples. The *n*-tuples that have their individual components belonging to the same classes are grouped in the same *n*-tuple class.

As an example, for the NAND2 gate presented in Fig. 4.11 with three sensitive zones, the initial currents data base generated by DASIE tool contained 88,755 triplets of currents. Three types of current shapes were defined. Then, we created 16 current classes for each shape. Finally, we used these current classes to group the *n*-tuples. Thanks to this process, the number of currents was reduced to 3,399, which represents only 3.83% of the total number.

4.4.2 List of Sensitive Nodes

To illustrate the analysis, we have considered an example of a NAND2 gate. Figure 4.16 shows the three sensitive zones for this gate at layout (a) and schematic (b) levels identified by TMC DASIE tool.

Fault injection of multiple transient currents is performed in the identified sensitive nodes. For example, if zone 1, zone 2, and zone 3 are concerned, for a specific neutron of a given energy, the three currents obtained by TMC DASIE will be injected simultaneously in the three sensitive zones, regardless their biasing conditions. In the case of heavy ions, only blocked drains are considered as sensitive nodes; therefore, it is important to identify circuit input vectors that will block a drain and perform transient fault injection only for that specific input combination. This is done by performing a normal simulation with all possible input combinations and considering the voltage value of the gate of each transistor to determine if its drain is blocked. Thus, for each input combination, the list of the blocked drains is created.

As an example, for heavy ions sensitivity analysis, the blocked (thus sensitive) zones for NAND2 gate are shown in the Table 4.3.

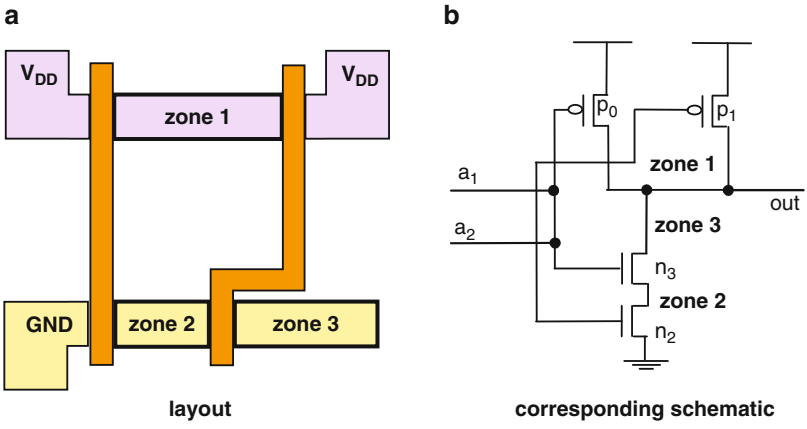


Fig. 4.16 Sensitive zones for the NAND2 gate

Table 4.3 Blocked sensitive zones for NAND2

Input combination		Blocked zones
$a1$	$a2$	
0	0	Zones 2 and 3
0	1	Zone 3
1	0	Zone 2
1	1	Zone 1

Also, in this stage, we identify for each zone the type of the corresponding transistor PMOS or NMOS to establish the sign of the current to be injected.

4.4.3 Automated Multiple Transient Currents Simulation

Thanks to the automated simulation, we could simulate thousands of n-tuple of currents for all the possible input combinations of complex logic gates in a single simulation session.

The simulation algorithm is presented below

```
for each analyzed design
  add stimuli and fault sources in the netlist
for each n-tuple of currents
    for each input combination
              multiple injection in sensitive zones
      save outputs
```

Figure 4.17 presents the automated simulation process. Different stages of netlist processing are performed before the effective simulations. First a golden simulation is performed, where only the input stimuli are added. Then, current sources are added. For each n-tuple current simulation, a new netlist is generated by replacing in the golden netlist the template fault sources with the actual ones.

4.4.4 Results Analysis

For each fault injection, the voltage pulse at the output of the cell is stored in PWL format, thus allowing an easy comparison between golden results and fault injection simulation results.

During the comparison phase, the duration of the voltage pulse produced at the cell output is measured at $V_{DD}/2$ (see Fig. 4.18).

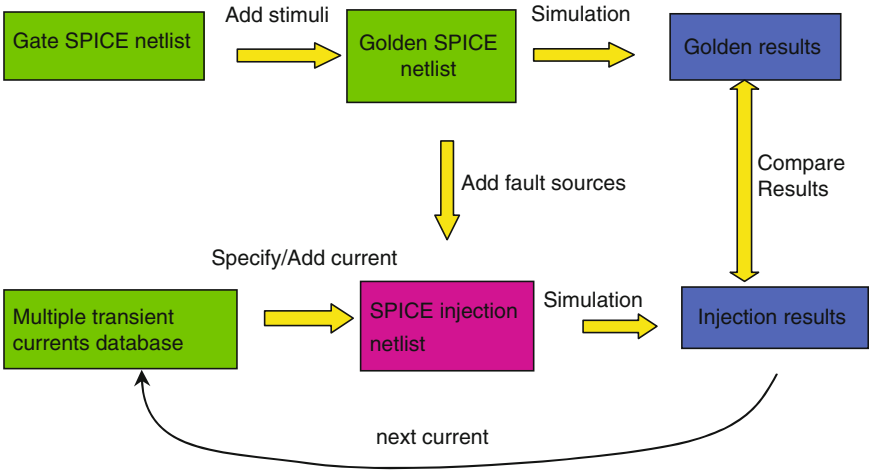


Fig. 4.17 Automated simulation methodology

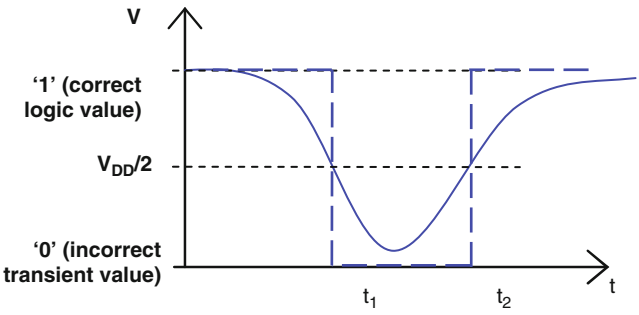


Fig. 4.18 Transient fault duration measurement

In fact, the duration of transients is computed for each input combination and each fault injection simulation.

4.4.5 Example of Application on an Inverter Case

For simplicity, the simulation mechanism for an inverter is shown. This cell contains only two transistors. Thus, two sensitive zones will be considered as injection points: the PMOS transistor drain and the NMOS transistor drain. For this gate, there is one input (two input values) and one output.

The currents are injected only on the sensitive zones of the blocked transistors, and so there are two possible injection simulations (see Fig. 4.19)

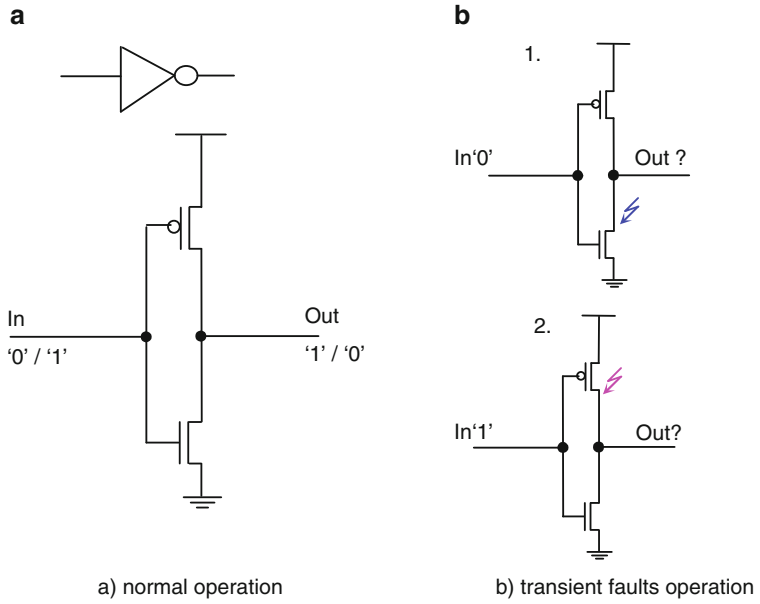


Fig. 4.19 Inverter behavior situations

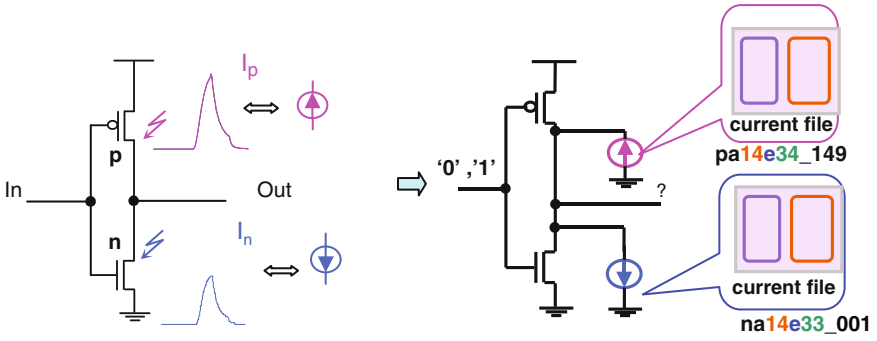


Fig. 4.20 Transient fault implementation

1. In = “0”, injection in NMOS drain
2. In = “1”, injection in PMOS drain

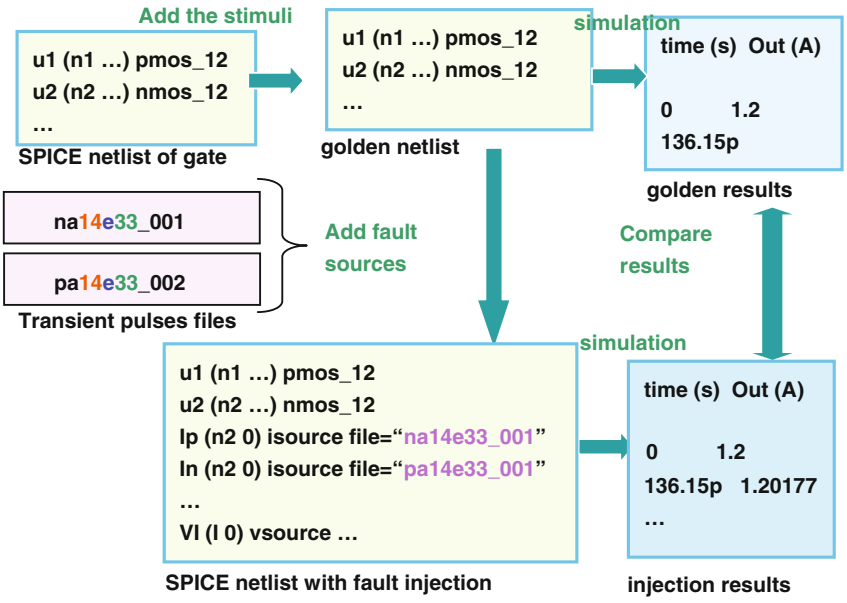
The transient faults are injected as current sources connected to each sensitive zone as shown in Fig. 4.20, where na14e33_001 and pa14e34_149 are SPICE PWL files describing the transient current.

The simulation algorithm is detailed below:

launch a simulation →

```
for each cell to analyze
    for each transient current n-tuple
        for each input combination
            for each transistor P
            {
                inject Ip in the drain of the transistor
                get outputs
            }
            for each transistor N
            {
                inject In in the drain of the transistor
                get outputs
            }
```

The whole simulation mechanism is the following



4.4.6 *Multiple Transient Fault Injection Results*

In the following, we present some results obtained after simulating a few combinational and sequential logic cells (90-nm technology) with multiple transient injections. The currents used for injection were produced by neutrons with energies

Table 4.4 Gate characteristics and simulation results for combinational cells and 63 MeV neutron reaction

Gate parameter	INV0	NAND2	INV8	XNOR2	OAI8
Inputs	1	2	1	2	8
Outputs	1	1	1	1	1
Transistor count	2	4	2	10	12
Sensitive zones	2	3	4	12	12
Ratio of TF (%)	13.66	3.86	0.01	2.34	4.83
SER (cm^2/cell)	4.03E-14	1.32E-14	1.23E-16	2.07E-14	8.24E-14

ranging from 1 to 200 MeV. To compute the SER, we are interested only for the cases where the voltage pulse produced at the output of the cell has amplitude higher than $V_{DD}/2$. We present here the results for several gates: two inverters (INV0 and INV8), a NAND gate (NAND2), an XNOR gate, a more complex gate, an eight input OAI (OAI8), and a D-latch. The characteristics of combinational cells and the simulation results for these gates for 63-MeV neutrons reactions are presented in Table 4.4.

In Table 4.4, "inputs," "outputs," "Transistor count," and "sensitive zones" rows show respectively the number of gate inputs, gate outputs, number of transistors in the gate, and number of sensitive zones. The transient currents used for injection are produced by neutrons having 63 MeV. "Ratio of TF" row shows the ratio of injected currents that produce output voltage pulses, which at $V_{DD}/2$ have duration higher than the shorter logic transition time of the cell library (in practice, when simulating a netlist, we will consider the logic transition time of the fanout gates of the gate under simulation). The last row provides the computed SER (or the cross-section). To compute the SER, we multiply the ratio of injected currents that produce transient faults (TF) by the occurrence probability of the transient currents computed by TMC DASIE. For a NAND2 gate, only 3.86% of the transient currents injected have produced transient faults at the output of the gate. For a simple inverter this probability is higher. As a general rule, the ratio of currents that give transient faults decreases as the cell complexity increases (see also Figs. 4.21 and 4.22 where results of Inverter, NAND, XOR, and AOI gates are compared).

Figure 4.23 presents the probability distributions of the transient pulse durations occurring at the NAND gate output. The results are obtained by injection of multiple transient currents produced by neutrons with energy ranging from 1 to 200 MeV.

We observe that very few pulses have duration larger than 150 ps, and this happens for high energy neutrons, while most of the currents produced at the output of the gate have durations in the range of (50–150 ps).

Another type of analysis concerns the impact of multiple transient currents on a gate with respect to the input combinations. Considering as an example the multiple transient currents produced by neutrons having an energy of 63 MeV, we can observe that the input combination $(a1, a2) =$ "10" is the most sensitive one (Table 4.5). This information is very important for system soft error analysis. The same type of analysis can be realized for the ensemble of neutron energies considered (see Table 4.6, for atmospheric spectrum).

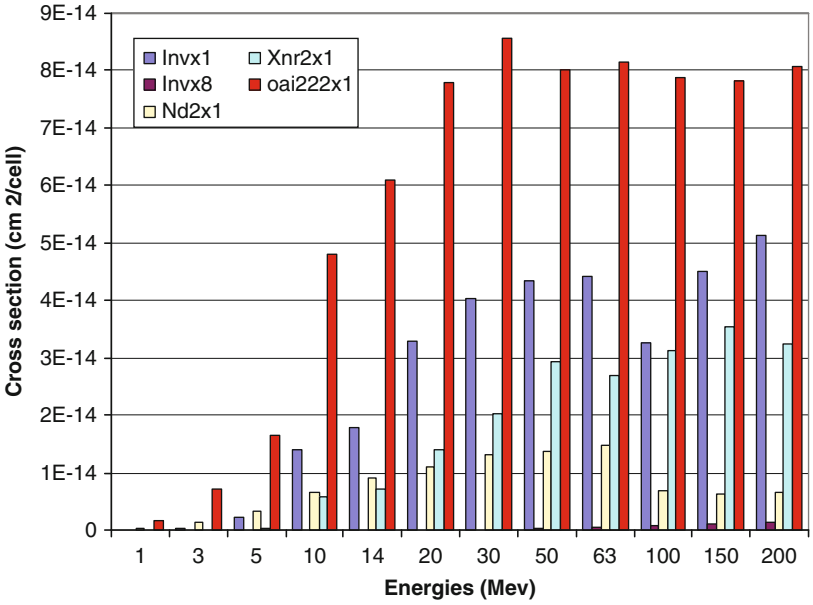


Fig. 4.21 Cross-sections for the combinational cells

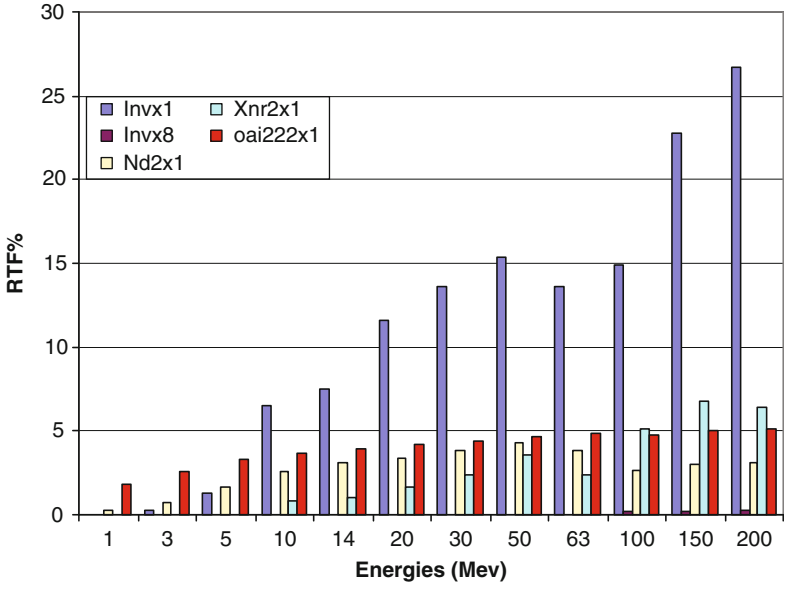


Fig. 4.22 Ratio of TF for the combinational cells

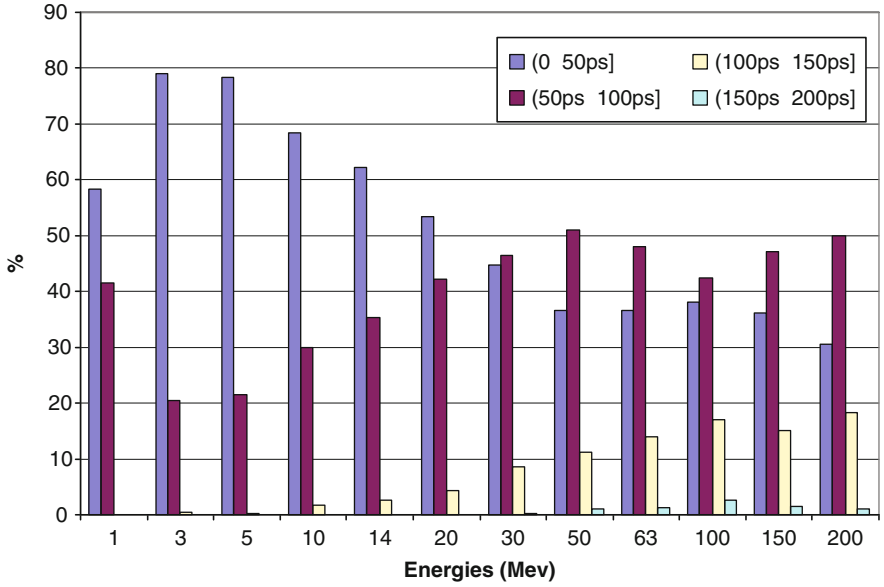


Fig. 4.23 Transient fault duration relative probability distribution for NAND2 gate

Table 4.5 Failure rate probability for different input combinations and currents produced by 63 MeV neutrons (NAND2)

a_1	a_2	Failure rate (%)
0	0	6.97E-14
0	1	7.61E-14
1	0	1E-13
1	1	6.31E-14

Table 4.6 Failure rate probability for different input combinations and all neutron energies (1–200 MeV, atmospheric spectrum), for NAND2

a_1	a_2	Failure rate (%)
0	0	6.43E-13
0	1	6.98E-13
1	0	9.28E-13
1	1	5.70E-13

4.5 Conclusion

In this chapter, we presented a methodology for estimating probability of failures in combinational and sequential cells induced by neutrons and protons. Contrary to previous approaches, we have considered the realistic case of charge sharing, where charge can be collected by several sensitive nodes of a cell. It results in

multiple transient current pulses induced in a cell by a single event. The approach includes Monte Carlo-based selection of nuclear reactions (neutrons or protons with silicon), and physical simulation of the carriers transport in the device based on a diffusion–collection model. An automated SPICE level simulation tool is presented and used to simulate the obtained realistic current pulses, which in most complex gates are multiple. Statistics can be obtained regarding the failure probability of each gate, considering different parameters (particle energy, input combination, duration of the transient pulse required for inducing a fault). The results can be further exploited at logic and system levels for performing soft error vulnerability analysis and soft-error mitigation.

Acknowledgments The authors would like to thank Antonin Bougerol, Claudia Rusu, Jean-Marie Palau, Frederic Wrobel, Remi Gaillard, Guillaume Hubert, and et Seddik Benhammadi for their important contributions in the theory and simulation methodology presented in this chapter.

References

1. A. Taber and E. Normand, “Single event upsets in avionics”, IEEE Trans. Nucl. Sci., vol. 40, no. 6, pp. 1484–1490, 1993.
2. C. A. Gossett et al., “Single event phenomena in atmospheric neutron environments”, IEEE Trans. Nucl. Sci., vol. 40, no. 6, pp. 1845, 1993.
3. J. F. Ziegler and G. R. Srinivasan, “Terrestrial Cosmic Rays and Soft Errors”, IBM: J. Res. Dev., pp. 19–39, January 1996.
4. E. Normand, “Single event upset at ground level”, IEEE Trans. Nucl. Sci., vol. 43, pp. 2742–2750, 1996.
5. E. Normand, “Extensions of the burst generation rate method for wider application to proton/neutron induced single event effects”, IEEE Trans. Nucl. Sci., vol. 45, no. 6, pp. 2904–2914, 1998.
6. P. Calvel et al., “An empirical model for predicting proton induced upset”, IEEE Trans. Nucl. Sci., vol. 43, no. 6, pp. 2827, 1996.
7. L. D. Edmonds, “Proton SEU cross sections derived from heavy ion test data”, IEEE Trans. Nucl. Sci., vol. 47, no. 5, pp. 1713–1728, 2000.
8. C. Vial et al., “A new approach for the prediction of the neutron-induced SEU rate”, IEEE Trans. Nucl. Sci., vol. 45, pp. 2915–2920, 1998.
9. G. Hubert et al., “Detailed analysis of secondary ions’ effect for the calculation of neutron-induced SER in SRAMs”, IEEE Trans. Nucl. Sci., vol. 48, no. 6, pp. 1953–1959, 2001.
10. G. Hubert et al., “Review of DASIE Family Code: Contribution to SEU/MBU Understanding”, 11th IEEE International On-Line Testing Symposium 2005, Saint-Raphael, France.
11. G. Hubert et al., “Simplified Monte Carlo DASIE method dedicated to the MBU understanding and characterization in integrated devices”, NSREC 2005, Seattle, USA.
12. Y. Yahagi et al., “Self-consistent integrated system for susceptibility to terrestrial neutron induced soft-error of sub-quarter micron memory devices”, Integrated Reliability Workshop Final Report, 2002. IEEE International, 21–24 Oct. 2002, pp. 143–146.
13. H. H. K. Tang et al., “SEMM-2: a modeling system for single event upset analysis”, IEEE Trans. Nucl. Sci., vol. 51, no. 6, Part 2, pp. 3342–3348, 2004.
14. P. C. Murley and G. R. Srinivasan. “Soft-error Monte Carlo modeling program, SEMM”. IBM J. Res. Dev., vol. 40, no. 1, pp. 109–118, 1996.

15. F. Wrobel et al., “Use of Nuclear Codes for Neutron-Induced Reactions in Microelectronics”, 11th IEEE International On-Line Testing Symposium 2005, Saint-Raphael, France.
16. G. Hubert et al., “Study of basic mechanisms induced by an ionizing particle on simple structures”, IEEE Trans. Nucl. Sci., vol. 47, pp. 519–526, 2000.
17. PH. Roche et al., “Determination on key parameters for SEU using full cell 3-D SRAM simulations”, IEEE Trans. Nucl. Sci., vol. 46, pp. 1354–1362, 1999.
18. J.-M. Palau et al., “Device simulation study of the SEU sensitivity of SRAMs to internal ion tracks generated by nuclear reactions”, IEEE Trans. Nucl. Sci., vol. 48, no. 2, pp. 225–231, 2001.
19. N. Seifert et al., “Radiation-induced soft error rates of advanced CMOS bulk devices”, 44th Annual International Reliability Physics Symposium, 2006, IEEE, pp. 217–225.
20. G. C. Messenger, “Collection of charge on junction nodes from ion tracks”, IEEE Trans. Nucl. Sci., vol. 29, no. 6, pp. 2024–2031, 1982.

Chapter 5
Circuit and System Level Single-Event Effects Modeling and Simulation

Dan Alexandrescu

This chapter covers the behavior of complex circuits and systems in the presence of single-event effects, and the transformation of the related faults to errors and errors to functional failures. In addition, an overview of practical methods and techniques for single-event effects analysis is presented, attempting to help the reliability engineers to cope with the single-event rate constraints of modern designs.

5.1 Introduction

The technological scaling allows building increasingly complex electronic devices integrating more and more functions. This evolution is not free of problems. As presented in the previous chapters, single-event effects (SEEs) represent the most important cause of reliability reduction in nanometric CMOS [1].

Originally, SEE issues were limited to electronic systems deployed in radiation hostile environments like space [2] and nuclear plants. Other applications were affected by SEEs much later, starting with some safety critical systems, such as implantable medical devices [3], and followed by a growing number of application domains as nanometric scaling was accompanied with increasing sensitivity to SEEs. As the failure in time (FIT) per chip increases (due to the combined effects of device size and voltage level reduction, growing complexity, and increasing integration levels), an increasing number of systems cannot meet reliability levels acceptable for their target application unless special care is taken during their design. Established solutions from space and military applications are available, however at a high cost, which may drastically limit their suitability to commercial applications.

D. Alexandrescu (✉)
VP Engineering, Board Member iRoC Technologies, WTC Po Box 1510, 38025 Grenoble, France
e-mail: dan.alexandrescu@iroctech.com

M. Nicolaidis (ed.), *Soft Errors in Modern Electronic Systems*,
Frontiers in Electronic Testing 41, DOI 10.1007/978-1-4419-6993-4_5,
© Springer Science+Business Media, LLC 2011

Multiple approaches are possible: a less sensitive process such as SOI, hardened cells, circuit and system-error mitigation techniques, etc. Most of these solutions come at some added costs; thus, some trade-offs must be found between error handling capability and cost.

Memory devices are almost always the first circuits to be implemented in a new process. Their highly regular structure makes them perfect candidates and a highly effective benchmark for any technological evolution. In addition, there are some efficient error-mitigation techniques such as Error Detecting and Correcting (EDAC) codes enabling reduction of the effects of soft errors (SE_S) at low cost. Thus, protecting memory blocks is an effective and affordable way to reduce the soft-error sensitivity of a design.

In contrast, logic networks have a much more complex structure that allows the SEEs to manifest in very diverse ways with varying levels of criticality. In opposition to soft errors in memories, where it is relatively easy to constraint errors to affect a single bit per memory word, it is practically impossible to impose such constraints in logic. Furthermore, implementing code prediction circuitry in logic is expensive. Thus, protecting logic blocks is a difficult task and may introduce important penalties in terms of area, speed, and power. To determine the best compromise between protection level and costs, the first task that must be accomplished consists in estimating the SEE sensitivity of the circuit. As soon as soft-error rate (SER) figures are associated with the various blocks of the circuit, the designer can decide which blocks need protection, and what mitigation techniques are most adapted.

However, evaluating the SER of a complex circuit with respect to SEEs is not an easy task. Intrinsic (raw) cell sensitivity figures must be provided as a starting point. This is a challenge by itself [4–6]. Then, the circuit description must be provided in some format and used to determine its SER. Thus, the SER analysis should be able to work with different circuit representations: architecture, block-level diagram, a high-level Hardware Description Language, RTL, gate-level netlist, transistor, etc.

An accurate analysis requires working on a circuit description that is closest to the final implementation. However, this is not always practical or possible. An SEE expert will have to work closely with the design engineers in order to better understand the behavior of the circuit and also to propose improvements and error-mitigation techniques that have a better chance at implementation. Thus, it will be most beneficial to use the same files, techniques, and tools as those used by the design engineers. RTL files and a design simulation and validation environment will be critical for evaluating the contribution of single-event upsets (SEU) to SER at the SoC level. The contribution to the SER of single-event transients (SET) cannot be done at this level. Structural gate-level netlist and timing information will be needed in this case. Lastly, the usage of the circuit in the actual working conditions must also be taken in account, since different workloads/applications may behave differently in the presence of perturbations.

However, evaluating these effects is just the beginning of the process. Following the analysis [7], the circuit will have to be protected versus SEEs. Several techniques are possible and the SEE expert must accompany the designers in devising the best protection methodology for the given circuit/application.

Given the complexity of the circuit and the task at hand, SEE-specific EDA tools can be very helpful. In addition, SEE constraints can be added very early in the design flow to better integrate and respect them during the design phase. It should be quite beneficial if the standard design tools are able to consider and optimize SER performances in addition to "classic" figures such as speed, area, power consumption, yield, etc. Adequate EDA tools and methodologies can help tremendously to minimize the SEEs' impact, to avoid unnecessary design iterations, and to diminish the time-to-market.

Dealing with all these subjects, this chapter hopes to improve the SEE awareness during the design flow and to offer practical solutions, helping both SER analysis and improvement efforts. We will start by looking at the most straightforward approach consisting of simulating SET and SEU faults in netlist and RTL circuits. Improvements to this approach will be proposed aiming at reducing the analysis time and/or a more complete coverage. Inspired by the simulation approach, we will then present faster, static approaches that help the designer to analyze complex ASICs in reduced time with much simplified setup.

5.2 Definition of Scope and Objectives

5.2.1 Single-Event Effects Models and Metrics

5.2.1.1 The Single-Event Transient

The SET is a direct consequence of an SEE. The production mechanism (Fig. 5.1a) and its manifestation at the output of the affected cell (Fig. 5.1b) have been presented extensively in the previous chapters and in the literature [8–15].

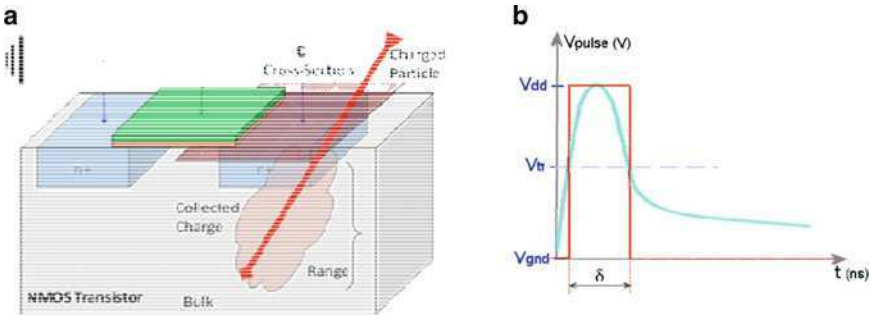


Fig. 5.1 Single-event transient – production and modeling. (**a**) Interaction of a charged particle with the transistor. (**b**) SET: Analog transient and logic model

In the context of this chapter, the most effective way to represent the SET consists in transient logic fault that flips the logic value of the affected signal during a time interval. This type of phenomenon appears with a very low probability for a natural working environment. We call this probability as the SET SER and can be expressed in FIT. One FIT corresponds to one SET during a billion working hours. In memories, we usually compute the FIT for a megabit. In the following, we consider the SET FIT for 2^{20} logic cells (a MegaCell).[1]

During the analysis, we only consider transient pulses that have sufficient voltage amplitude to switch the transistors of the fan-out cell(s) ($V_{SET} > V_{THRESHOLD}$). Additionally, the transient pulse duration (PW – pulse width) should be large enough to be able to propagate at least through a few levels of combinational cells. A necessary but not sufficient condition is that the PW is larger than the delays of the following cells. These conditions are equivalent to considering the initial phenomenon and applying an electrical derating (EDR) factor. In this chapter, we consider that all SETs are purely logical (already derated). In this context, the model that we can use is a short logical glitch (transient fault), with a known duration – pulse width (Fig. 5.1b).

However, not all events are equal; the SET PWs may vary considerably from one event to the other. The electrical environment (neighborhood) of the affected cell has a very strong impact. Thus, several factors may have to be considered:

- *The state of the cell*. The actual electrical configuration at the impact time determines the sensitive volumes of the various transistors of the cell. Accordingly, the charge deposited by the particle will cause a perturbation or not, with varying intensity.
- *Supply voltage*. A lower supply voltage will make the cell more sensitive to SEEs and vice versa.
- *The capacitive load on the charge collection node*. The collection of the deposited charge is dependent on the capacitance seen at the collection site, which is generally dependent on the cell output load. Furthermore, the capacitive load of the cell output has an impact on the voltage pulse produced on this output for a given current pulse generated on the charge collection node(s).
- *Threshold voltages of the downstream cell*. The actual shape of the analog SET pulse has specific slopes for the two signal transitions. The analog transients may be quite short and of lesser amplitude than normal signal transitions. The cells connected at the affected output will switch according to their own threshold

[1]We have improperly adopted the notion of SER and FIT (Chap. 3) to this context. SER actually means Soft Error Rate and should measure the number of genuine soft errors, that is, errors whose effects on the circuit operation are only determined by the function of the circuit. But an SET is a transient perturbation of a combinational circuit node, whose effect on the circuit depends on its occurrence time and its duration, as well as the structural implementation (net-list) and the temporal characteristics of the circuit. This is also the case for an SEU affecting a sequential element, as its impact on the circuit operation depends on the time of its occurrence and on the temporal characteristics of the circuit. Remember also that FIT means failure in time, while an SET or an SEU does not necessarily induce a failure.

voltage for the input affected by the transient. One satisfactory approximation is that the threshold voltage is considered equal across all cells ($V_{dd}/2$).

- *Short/feeble analog transient propagation*. On the output of the downstream cell, we may witness either a definite and clear logic transient with fast slopes (regeneration effect) or a further attenuating transient that will quickly disappear during the propagation through the logic combinational network. These effects are difficult to analyze using a strict logic analysis or simulation. The most precise solution consists in using analog (SPICE) simulation for evaluating the effects of the SET on the neighborhood of the affected cell, until we can clearly observe the normalization or the masking of the fault. One practical approach is to consider that transient pulses with long enough duration are propagated correctly, and the usual timing analysis or simulation methods are satisfactory for dealing with them (Logic SET). Shorter pulses will also be taken into account, but any corresponding results should be seen as worst case. What exactly are the criteria for deciding whether the pulse is long enough or not depends on the actual cell implementation. One reasonable assumption is to consider the threshold value as three times the delay of the basic inverter. Of course, this assumption will be optimistic for some nodes and pessimistic for other nodes, but will give good results after the optimistic and pessimistic cases are averaged over a large number of transient simulations.

Beyond the above simplifications, a more accurate approach will use SPICE simulations to determine the voltage pulses created at the output of each cell for a selected set of output loads. These pulses are stored in a pulse library. During circuit simulation, the pulse produced on the output of a cell for the actual load is computed from the stored pulses using a linear extrapolation. Similarly, we can simulate the propagation, through each cell of a library, of a few pulses having selected durations, and store the results in a pulse-propagation library. Then, during circuit simulation, the propagation of an actual pulse through a cell will be extrapolated from this library using linear approximation.

Regardless of the approach used for evaluating the SET SER of a combinational cell, the SET SER of the cell for a given output load is described by a table containing the FIT values for different PWs and for each state (i.e., input value) of the cell. An example of an SET SER characterization table for an inverter ($\times 1$ 90 nm inverter), a given load (7 pF), working in a natural environment (atmospheric neutrons, sea level) is shown below:

PW (ps)	10	20	50	100	150	200	250	300	350
SER (FITs) In = 1 Out = 0	200	175	100	50	10	–	–	–	–
SER (FITs) In = 0 Out = 1	300	250	200	150	100	50	25	10	–
SET type	Analog	Analog	Logic	Logic	Logic	Logic	Logic	Logic	–

The distribution of the transient pulses can be obtained by radiation experiments over specific structures able to measure the PWs [14, 16], or by using dedicated tools. One possibility is to use TCAD simulation, but the task is intractable as each

simulation at the TCAD level may take several hours, while there are a huge number of particle types, energies, particle interaction sites, and particle trajectories to simulate. Fast tools working with simpler physical models than TCAD, such as TMC-DASIE presented in Chap. 4, or using preliminary TCAD simulation to create transient pulse models taking into account detailed process information such as TFIT [17, 18], are more suitable for this task.

5.2.1.2 The Single-Event Upset

The production mechanism of an SEU in a sequential cell consists in the propagation of an SET affecting a node of the cell through the cell's memorization loop [19, 20]. As a consequence of this propagation, the stored value is flipped and kept in the loop until the next latching event (e.g., the latching edge of the clock). It can be safely modeled as a logic fault since the output of the cell changes firmly.

Latches (Fig. 5.2a) are transparent for one phase of the clock and closed for the opposite phase. During the transparent phase (Fig. 5.2b), their behavior is purely combinational and an SEU is not possible. The cell is still affected by SETs that can be propagated to the next stage of sequential elements, and the reasoning from the previous paragraph is applicable. During the closed (memorization) state (Fig. 5.2c), the latch memorization loop is sensitive to an SEU. In this case, the only parameter of interests is the SEU SER (FIT) for the different states of the cell.

The most common implementation of flip-flop (Fig. 5.3a) consists of two latches (master and slave).

During one phase of the clock, the master latch is closed (Fig. 5.3b) and the slave is transparent, propagating the value of the master to the output. In the case of standard flip-flops, the memorizing is triggered by the positive edge of the clock and the master stores the value during the first clock phase. An SEU may affect the master latch and the slave will immediately propagate the value to the flip-flop output. Since in the second phase the slave will latch the incorrect value, the cell output will continue to be affected until the next clock activation. The most important parameter is the event rate (SER) or the sensitivity of the master latch.

In the case where the SEU affects the slave latch (only possible during the second phase of the clock), the fault will obviously propagate to the output. The

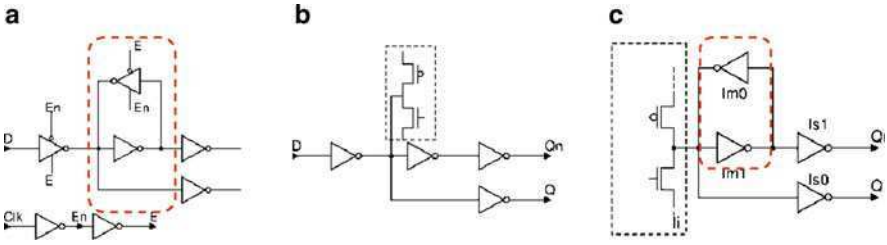


Fig. 5.2 Latch configurations. (**a**) Equivalent latch structure. (**b**) Open (transparent) latch. (**c**) Closed (memorizing) latch

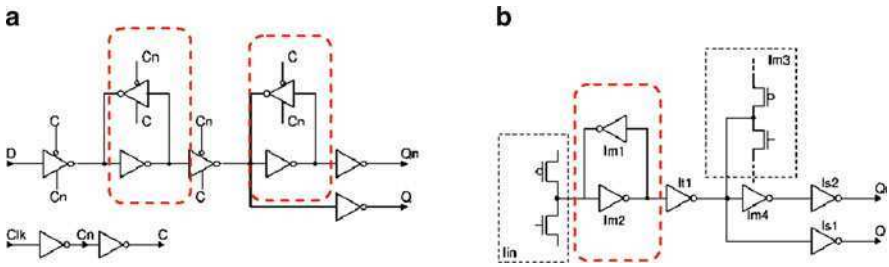


Fig. 5.3 Flip-Flop configurations. (**a**) Equivalent Flip-Flop structure with the master and slave latches. (**b**) Master is memorizing, slave is transparent

most important parameter is again the event rate (SER) of the slave latch. Please note that the sensitivities of the master and the slave latches may be different. Also, as the SEU in the master occurs earlier, its impact may be much more critical from a timing perspective than that of the later arriving SEU in the slave latch. These considerations will be discussed in a following sub-chapter.

Single-Event Transients may also affect latches and flip-flops during the transparent phase of the master latch or the slave latch of a flip-flop, or directly in the output stages. Their effects may be analyzed similarly to an SET in a combinational cell. We can discard the SET that appears in the memorizing phase and does not cause any SEU, since these effects are very weak and will not propagate as transients through the combinational circuit. One possible simplification that can be reasonably applied for current technologies (65–130 nm) is that the SET on the output of a sequential cell is strongly de-rated from an electrical and timing perspective. Thus, the SEU contribution on the SER of the sequential cell is much more important than the SET contribution.

SER data for sequential cells can be obtained by means of radiation tests or dedicated tools such as TMC-DASIE [21] or TFIT [17, 18].

5.2.1.3 The Single Bit Upset, Multiple Cell Upset, and Multiple Bit Upset in Memory Blocks

Single bit upsets (SBUs) are the single bit flips induced in a memory by SEEs [22–33]. The interaction of an ionizing particle with the memory is obviously dependent on the type of the memory. As an example, the production mechanism of the SBU in an SRAM device (Fig. 5.4a) is similar to the SEU mechanism described earlier for sequential cells. For the SRAM cell, this mechanism is illustrated in Fig. 5.4b. For DRAMs, the production mechanism is totally different. The deposited charge is injected in the storage capacitor, and may flip the value stored in the capacitor. The manifestation of the SBU is a single faulty bit in the data read from the affected address.

A single particle or several secondary particles produced by the interaction of an energetic neutron or proton with the devices' atoms may affect several neighborhood

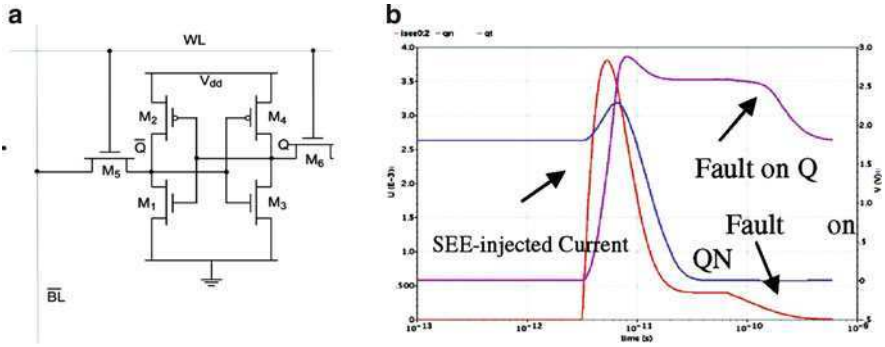


Fig. 5.4 SEEs on a Standard 6T Memory. (**a**) Equivalent structure of a 6T SRAM. (**b**) The effect of an SEE on the SRAM cell

cells at the same time, causing a multiple cell upset (MCU) [34, 35]. Thus, several bits of the memory will be affected by the fault. This is an increasingly critical problem for latest processes (45 and 32 nm at the time of writing this book). As an example, the Fig. 5.5 shows the relative distribution of MCUs versus SBUs for a 45-nm process. The external manifestation of the MCU, as seen from the read port, depends on the internal organization of the memory. If physically neighborhood cells belong to the same logical word, then MBUs are produced. Most "sanely" designed memories have a sparse topology: bits belonging to the same logical word are not placed physically close. However, MCUs with large multiplicity are still possible, although with a low probability. Thus, even if the memory is designed with a carefully selected cell interleaving, MBUs are still possible, although with a very low FIT.

The MCU/MBU analysis is particularly interesting when considering the eventual error protection mechanisms. SBUs are corrected by the most common error-mitigation technique – the Single Error Correcting, Double Error Detecting (SECDED) codes, such as the Hamming code. Thus, the SBUs in SECDED protected memories will not need particular care. However, the MBUs will not be corrected by this code, causing further errors in the circuit. The actual response of the circuit will have to be analyzed.

In any case, the model that we can use when dealing with consequences of an SBU or a MBU is a logic value flip on one or more bits in the value read from the memory, as in the following example:

Fault type	Correct data (binary/hex)	Faulty data (binary/hex)
0×000002	0b01010101/0×55	0×01110101/0×65
0×000003	0b01010101/0×55	0×00110101/0×35

The occurrence probability of an SBU/MBU can be expressed in FITs per megabit. One FIT is one SBU/MBU during a billion working hours for a megabit of memory $= 2^{20}$ (1,048,576) bits.

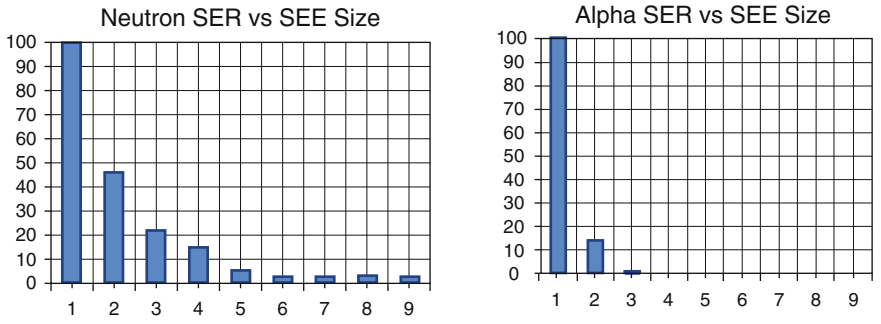


Fig. 5.5 Relative SBU/MCU distributions for a 45-nm process (averaged from iRoC technologies radiation testing results over a large number of memories)

SER figures for SBUs, MCUs, and MBUs can be obtained by means of radiation tests or dedicated tools such as TMC-DASIE [21] or TFIT [17, 18].

5.2.1.4 The Single-Event Functional Interrupt for Memories

The single event functional interrupts (SEFIs) are a class of failures where the addressing/decoding/control logic of a memory is affected by a single event [36]. The manifestations can be very varied: address ranges replying with incorrect data, semi-permanent stuck bits on data read from the memory, device not responding/not outputting data, etc. Thus, the exact error model to be used should be chosen according to the type of behavior we want to simulate.

5.2.1.5 The Soft Error

Both transients (SETs) and upsets (SEUs) are the direct results of SEEs. As an SEE may appear at any time during the clock period, a first condition that determines the effect of the SET/SEU on the circuit operation concerns the relationship between the instant it reaches a sequential element and the clock edges. Indeed, the transient pulse has a short lifetime: if it is not latched in a sequential cell or a memory element, it will disappear from the circuit without traces. For example, an SET arriving on the data input of a flip-flop should affect this input during the *latching window* of the flip-flop (the time interval during which a transient should reach the input of a sequential cell in order to be latched). It corresponds to the setup and the hold time. According to the delay between the node of its occurrence and the sequential elements, an upset should occur early enough in the clock period in order to reach these elements before the end of this window and late enough in order to reach them after the beginning of this window.

In logic circuits, we will define as soft error (SE) an error produced by an SET affecting a combinational cell or an SEU affecting a sequential cell, and latched by the next stage of sequential/memory cells. Thus, an SET or an SEU is transformed

into an SE after it is propagated through the subsequent combinational logic and is latched by the subsequent stage of sequential/memory cells. It may seem paradoxical that an SEU occurring in a sequential cell is not considered as an SE, though it is latched by this element. So, a clarification is needed. In fact, we consider as SE, an error induced by an SEE and whose impact on the system operation does not depend on the temporal characteristics of the fault (duration or instant of its occurrence), on the temporal characteristics of the circuit, or on the structural characteristics such as the netlist implementation of a logic function, but only on circuit functionality. In this regard, an SEU in a sequential cell cannot be considered as an SE, because an SEU can affect a sequential element at any time during the clock cycle and may or may not affect the subsequent sequential elements according to its time of occurrence and the temporal characteristics of the circuit. From the above definition, an important difference between SEUs and SEs is that the later can be analyzed by a high level simulation, while this is not the case for the former.

SBUs/MBUS in memories can be considered as SEs, as they do not disappear at the end of the current clock cycle whatever their instant of occurrence and the temporal or structural characteristics of the design. Furthermore, there is a significant difference between errors affecting the logic and errors affecting the memories. SEs induced by SETs and SEUs modify the state of one or more sequential elements (latches or flip-flops) at the end of the clock cycle in which they occur. In the subsequent clock cycles, they will be propagated through the next circuit stages (or eventually masked). On the contrary, SBUs/MBUs will remain silent in the memory and will not be propagated through the logic until the next read of the affected memory cells. Also, they will never affect the computation if after their occurrence the first operation over the affected cells is a write.

5.2.2 The Functional Failure

We can provide clear and quite precise definitions for SEEs, SETs, SEUs, and SEs, whereas the Functional Failure is somewhat more difficult to define. The name suggests that the circuit or device failed to fill the function for which it was designed. However, there is a lot of wiggling room for how to qualify/define the consequence of the failure on the final application.

The failure mode definition should be left to whoever is competent or has the responsibility to perform this task. For each failure mode, the behavior of the circuit should be clearly described, allowing the SE expert to evaluate whether SEEs can cause each of them and determine the corresponding probabilities for a given working environment.

5.2.2.1 The Soft Error Rate

The SER represents the occurrence probability/rate of SE or SE-induced functional failures in a given working environment.

The SER can be expressed in FITs. One FIT is one SE or functional failure for a billion working hours.

As the SER (measured in FIT) can be given per bit; per megabit of memory; per MegaCell of logic gates; or per whole chip, board, or system, to avoid confusion, it is important to specify each time the amount of circuit for which a FIT figure is given. The FIT can also be given separately for SEs induced by SETs, SBUs, MBUs, MCUs, SEFIs, etc., or for a combination of them. This also has to be clearly indicated. Similarly it has to be indicated whether we provide an SER induced by alpha particles, by neutrons, etc., or by a combination of different kinds of particles.

For a complete circuit, the SER usually describes the occurrence probabilities of functional failures induced by all possible SEs affecting its constituent blocks (which is different from the sum of the FIT of these blocks). This can be extended to a whole board or system.

5.2.3 Circuit Representation and Abstraction Levels

5.2.3.1 Introduction

As stated earlier, SER analysis is most effective when used as an integral part of the design flow, using standard design files and tools. Of course, the primary concern should be the adequacy of any type of available information for the specific task of evaluating the effects of SEEs on complex circuits. Thus, the circuit representation could be subject to the following considerations:

- The description should be similar to the final implementation of the circuit from a structural perspective. The kind of SEEs affecting a cell is strongly dependent on the cell's structure. In addition, the characteristics of SETs are affected by the neighborhood (fan-out) of the source cell, since the characteristics of the transient pulse are affected by the capacity load of the cell's output.
- Timing information (delays of cells) is useful for evaluating the transient pulse deformation of SETs, induced during their propagation by unequal 0-to-1 and 1-to-0 intrinsic cell delay. In addition, timing information (path delay/slack, setup/hold time) is useful for analyzing the propagation of SETs/SEUs; determining the instant they reach the sequential elements and the width of the opportunity window (the time interval during which a transient should occur on a circuit node in order to be captured by a subsequent sequential cell); and calculating their latching probability (i.e., the probability that the SET/SEU is captured by a sequential cell and thus transformed into an SE).
- Higher level circuit representations (such as register transfer level (RTL)/block/architecture) can be still useful for determining the SER figures, but the estimation will be intrinsically less precise when the abstraction level is further away from the final implementation. Still, this kind of representation is critical for

SER budgetary purposes very early in the design flow and can also be used for deciding on mitigation measures.

5.2.3.2 Gate-Level Netlist Circuit Description

Based on these considerations, a gate-level netlist accompanied by timing information (e.g. a SDF – Standard Delay Format file) is well adapted to the purpose of performing SER analysis and can be obtained fairly easily at the later stages of the design flow. The netlist can be generated easily by a synthesis tool, thus allowing an early analysis. Timing information is available with increasing accuracy at different instants during the design flow: post-synthesis, post-layout, etc. Fast prototyping platforms or Physically Knowledgeable Synthesis may be helpful in providing earlier timing and more accurate timing information.

Specific SER improvements at the gate-level netlist/backend are possible: various In-Place-Optimization methods, implementing critical nodes in the netlist by using cells with low SER or hardened equivalents, TMR on selected cells/blocks, etc. These modifications are performed on the post-synthesis circuit, and may be considered as low level as the function of the circuit is not modified.

5.2.3.3 RTL/HLS Circuit Description

SER analysis may be required earlier in the design flow. The designer of a specific block may want to quickly assess the sensitivity of his block as soon as the (usually high-level) description is ready. Thus, waiting for a long logic synthesis flow that integrates contributions from other designers and teams is not always possible. Even if a partial synthesis is possible, this particular designer feels that he is most comfortable in working with his tools-of-trade: High-Level Synthesis (HLS), RTL/SystemC/SystemVerilog, functional simulators, etc. In addition, he may want to look for clever higher level error-mitigation techniques applied on the input circuit description. These improvements will then be carried over the lower level descriptions to the final implementation.

The quality of the SER analysis of a high-level circuit description is tributary to several factors:

Circuit structural information: It is quite important to understand how well the final implementation can be derived from the input description. Since the SEEs are strongly physical in nature, the event rate is strongly dependent on the actual cell types that will be used in the circuit. There are some types of representations where it is more difficult to retrieve structural data (SystemC) and others more appropriate for this purpose.

An RTL approach can be quite interesting, since memory elements (memory blocks and sequential cells) in the RTL description can be found relatively unchanged in the final implementation of the circuit. After some efforts, the contribution of the SEUs to the sensitivity of the circuit may be taken in account

with a satisfactory level of accuracy. The difficulty consists in identifying the RTL signals that will later be synthesized as memory elements. Syntactical rules or generic synthesis may help. Also important is the exact final cell type since different flip-flops or latches may have different SEU sensitivities. A manual, preliminary mapping can be performed in order to approximate the final circuit implementation. In addition, since most of the development effort is made at this level, any existing Design Validation (DV) environments can be successfully used to simulate the effect of the faults in the circuit. Such tools enable simulating a whole SoC (though for a moderate number of fault injections) and provide functional derating figures, which is not possible at lower description levels.

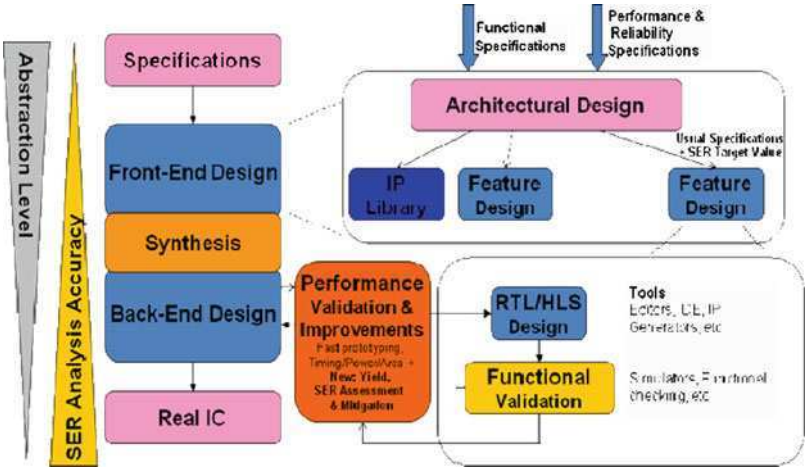
Fault model: The fault representation should match the physical phenomenon and be relevant to the chosen abstraction level. For example, for a RTL design, an SEU can be modeled as a logic fault (inversion of a logic value) that is applied on the affected register output until the next latching clock edge. However, SETs cannot be accurately described in this context since no timing information (delay) or combinational cell description is available at the RTL.

5.2.3.4 Architecture/Block-Level Circuit Description

Very early in the design flow, the system architect proposes a system architecture that will answer the initial specifications and constraints. In this architecture, the function and interface of the main blocks of the circuit/system are already identified. In addition, blocks also receive preliminary attributes that specify functional, performance, and reliability parameters. As the SE sensitivity has an impact on the reliability of the circuit, system architects started to associate SER targets to the various blocks of the design (budgeting) in order to respect an overall SER constraint. These constraints will be later propagated to the design team. The next step consists in checking the requirements against very preliminary SER figures. This initial assessment is based on assumptions and estimations about the structure and function of the circuit, previous experience, and ultimately some guesswork. If the actual SER estimations are much worse than the requirements, the system architect has the choice of letting the design team to deal with the problem (at an unknown cost and efficiency) or establish a set of recommendations concerning the error-mitigation techniques that will be applied on specific blocks or circuit features. As an example, memory blocks can be protected using ECC with a well-known area overhead and protection efficiency. This way, the system architect can revise the SER figures and include the effects of the error mitigation, both bad (area/time overheads) and good (decreased sensitivity).

5.2.3.5 Overview

Figure 5.6 illustrates how the SER analysis can be included in a typical design flow, by integrating SER requirements as a part of the overall reliability picture of the circuit:



Circuit		Architecture/Block	RTL/HLS	Gate-Level Netlist + Timing
Main interest		**SER Budgeting**	**SER Assessment for mitigation purposes**	**Accurate SER Assessment**
Design Flow		Very Early	Medium	Late
SEE Analysis Accuracy	SET	Poor/NA	Poor/NA	Very Good
	SEU	Satisfactory	Good	Very Good
	SE	Good	Very Good	Very Good
Positive aspects		Budgeting Ability to set SER targets Error mitigation strategy Implementation tech/lib may be decided early wrt SER	Adequate circuit representation Functional SER assessment and improvements Early cell/library choices Fast simulation, existing DV env.	Best accuracy SER of the actual implementation Specific SER optimizations
Negative aspects		Very approximate, final circuit may be widely different, No functional SER	Final circuit implem. not known Poor SET handling No timing considerations	Some SER improvements may arrive too late/unpractical Netlist & Timing files may be cumbersome to use (long simulations, big SDF files, etc)

Fig. 5.6 Soft error rate analysis during the design flow

5.3 SEE Analysis Methods and Concepts

5.3.1 Quantitative SEE Analysis

Single-Event Effects are a part of the overall reliability of the system/circuit. Indeed, SEEs as other types of faults may cause temporary or permanent loss of

function, or data corruption. "Classical" reliability notions may be applied and used by the reliability engineer to model and integrate the effects of the SEEs on the behavior of the circuit [5, 37].

The Mean Time Between Failures (MTBF) can be directly computed from the functional SER (FIT) rate. The Mean Time To Repair (MTTR) can be computed by computing the downtime associated with each type of SEE-induced failure. As an example, depending on the implementation, the ECC correction of an SBU has either no downtime (since most ECC implementations always detect and correct the error transparently in the same clock cycle) or one clock cycle delay (see Chap. 8). However, a MBU detected but not corrected by the ECC may have a stronger impact on the circuit, requiring a time-expensive higher level error handling protocol. Thus, the downtime associated with the different SEE-induced failures can be computed as for any other type of failures, and their effects can be evaluated quantitatively. For performing this computation, the SEE analysis can be broken down in to a series of steps, echoing the concepts from standard design flows. At each step, the user should employ the appropriate tools and concepts. More importantly, various steps could be tackled by expert users and the SER knowledge and expertise freely shared and exchanged during the SER analysis flow (Fig. 5.7).

The first step is to consider the physical phenomenon and the behavior of individual cells. This is a costly analysis and requires dedicated experts as well as time-consuming and costly test circuit design and radiation test experiments or dedicated tools [14, 16–18]. However, this is required only once for a given standard cell library. The outcome of this step is a rate/probability (SER) of occurrence of SET and SEU for combinational, sequential, and memory cells for a given environment. For the completeness sake, PW distributions should be provided for the transients. This is where the concept of "*Electrical Derating*" (EDR) is used to determine the percentage of SETs that will not have any impact on the circuit due to the low amplitude or duration of the transient pulse produced on the output of the affected cell. The results of the SER characterization of the cells will be associated with the standard cell library and will be used by the design engineers to evaluate the sensitivity of their design and to improve the circuit.

Thus, the second step is to consider the fault propagation from the output of the affected cell through the paths of the circuit [38, 39]. We must remember that an SEE may arrive at any instant during the clock period. The position of this instant

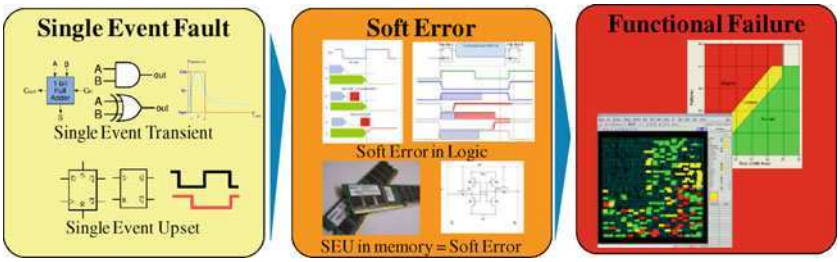


Fig. 5.7 Practical steps to evaluate single event effects on complex devices

with respect to the clock edge will determine if the fault will reach some sequential cells within their latching window and will be transformed into an error. This phenomenon is taken into account by the "*Temporal Derating*." Also, the fault may or may not be propagated from a logical perspective, as any other signal. This effect is taken into account by the "*Logic Derating*" [40]. This step determines the probability that the initial SEU or SET is propagated through the combinational logic and is captured by the sequential elements of the pipe-line stage in which it occurs, to produce an SE. Thus, this step analyzes the impact of the SEU or SET only during the clock cycle in which it occurs. A further analysis is required to determine the effects of the SE as it propagates through the whole design during the subsequent clock cycles.

Thus, the third step deals with the higher level consequences of the SE. The errors may be discarded (dropped) resulting in no effect on the circuit, may stay in the circuit (latent) without causing any observable failures during the considered time interval, or may cause failures with various degrees of criticality. The percentage of errors that become failures is represented by the "*Functional Derating*."

We would like to stress the difference between the "initial clock cycle" derating (electrical, temporal, and logic) and "many clock cycle" derating (functional derating). The propagation and latching of the fault in the initial clock cycle depend strongly on process, standard cell library, and circuit netlist implementation. Thus, starting from the same RTL/HLS/higher level representation, several implementations of the circuit are possible, with different inherent behavior in the presence of faults. However, the functional derating is invariable across different circuit implementations as long as the RTL description remains unchanged. Thus, electrical, temporal, and logic derating requires a circuit representation that matches the final implementation, although the functional derating accepts a higher level description.

In the following, we will present these derating factors and practical ways to compute them and take them in account according to the circuit purpose, implementation, and application.

5.3.2 Electrical Derating

As presented in Sect. 5.2.3, an SET can be modeled at the gate level as an inversion of a logic value during a specified duration. The duration should be chosen in such a way that the simulation of the fault will cause the same effects in the circuit as when a more precise simulation method (e.g. analog) is used. This may not be always possible since the circuit and signal propagation models that are available at the gate level may not be accurate enough for the simulation of short transient pulses with possible small amplitudes and durations. A more precise approach consists in using analog (SPICE) simulations to evaluate the SET propagation through the combinational cell network until we can safely decide whether the SET has disappeared or conclusively become wide and strong enough to be fully evaluated using only logic simulation.

Thus, EDR consists in evaluating the effective logic pulse that has the same effects in the circuit as the original analog pulse. The following aspects will have to be evaluated:

- *Capacitive load on the output of the affected cell*. The transient current pulse(s) resulting by the charge collection on the cell node(s) struck by the particle produces a voltage pulse on the cell output. This pulse can be computed by injecting the current pulse in the struck node(s) and simulating the cell at SPICE level. For a given current pulse, the characteristics of the voltage pulse occurring on the cell output depend on the cell load. Note also that the cell load may also have an impact on the transient current pulse. Indeed, if the struck node is isolated from the output load by a buffer, this load will not affect the charge collection and thus the transient current pulse. However, if no such buffer exists in the cell, the cell load has a strong impact on the current pulse. Tools such as TFIT [17, 18].
  - *Practical approach:* when characterizing the combinational cells at the physical level, consider several values for the load (e.g. $\times 1$, $\times 2$, $\times 3$, $\times 4$, $\times 8$, $\times 16$) and store the SET PW values in the SER database. Then during SET simulations, analyze the actual load on the cell and use linear extrapolation to determine the SET PW value from the stored data.
- *SET Slopes wrt. the linear behavior of the downstream cells*. In most cases, the SET on the output of the cell is not formatted (i.e., regenerated by an output buffer). Generally, the rising edge of the voltage pulse corresponds to a fast rising current pulse and will be faster than the falling edge, which corresponds to a slow falling current pulse. Since the slow slope will cause the downstream cell to work in the linear analog zone, there is no guarantee of linearization either. After the propagation through a few combinational cells, the SET will become either weaker and disappear or be formatted and become close to rectangular pulse.
  - *Practical approach:* simulate the transformation of a few slopes by each cell and store the result in the SER database. During circuit simulation, use the database to extrapolate the result for other slopes. Also, consider that the SET becomes rectangular after propagation through three logic levels to stop the extrapolations after the third level.
- *SET Amplitude wrt. the threshold voltages of the downstream cells*. The pulse seen by the cells connected to the output of the struck cell (fan-out cells) will be dependent on the relationship between the pulse amplitude and the input threshold voltages of these cells. Additionally, the non-rectangular SET shape will exacerbate the problem since the slow slopes will drive the following cells to switch at different instants according to their threshold voltages. This is not a very critical phenomenon and can be safely ignored most of the times.
  - *Practical approach:* considering that all cells have an identical $V_{threshold} = V_{dd}/2$ gives a good approximation. This rule also eliminates the consideration

of the impact of the pulse slope. It simplifies the computations significantly, while giving results of sufficient accuracy most of the time.

Single-Event Upsets affecting an inner memorization loop of a sequential cell do not require considering cell load, since these cells have an output buffer that isolates the memorization loop from the load. Thus, the SEU SER value will not depend on the fan-out of the sequential cell. Memory cells exhibit the same behavior when in storage (non-access) mode.

5.3.3 Temporal Derating

Temporal or timing derating deals with two different phenomena related to temporal aspects. The first consists in evaluating the effects of the SEE fault (SEU or SET) with respect to its occurrence time within the clock cycle. The second aspect only concerns SETs and treats the relevance of the pulse duration versus the clock period.

5.3.3.1 Temporal Derating for Single-Event Upsets

D-type flip-flops which constitute the backbone of most ASICs are usually controlled by a clock signal. The latching edge of the clock will trigger the flip-flop to memorize the value present in its data input. The data on the output of the flip-flop will change a short time afterwards and will act as an input to the combinational network. The outputs of the combinational network will be connected to another stage of flip-flops and so on. Combinational logic is sandwiched between stages of sequential logic. The clock period is larger than the delay of the combinational network placed between two stages of sequential logic, in order, to provide sufficient time to the circuit for computing new outputs in response to new inputs applied at each clock cycle. Some constraints such as the (t_{setup}, t_{hold}) parameters also have to be fulfilled for safe operations. Essentially, the data applied to the input of a flip-flop must be stable during a latching window determined by t_{setup} and t_{hold}.

An SEU may occur at any instant in the clock period. Following the event, the value stored in the flip-flop and applied to its output changes This value will be propagated through the logic network, as any other signal, and will reach the data input of the subsequent flip-flops at some time later. If the erroneous value arrives before the latching window of these flip-flops, the error will be present on the data input during the whole latching window, and will cause an error. The condition for this to happen is that the SEU occurs in the struck flip-flop early enough in the clock period (Fig. 5.8a). The opportunity window can be computed as a function of the delay (or slack) of the activated path between the source (struck) flip-flop and the destination flip-flops and the (t_{setup}, t_{hold}) parameters. If the SEU occurs after the

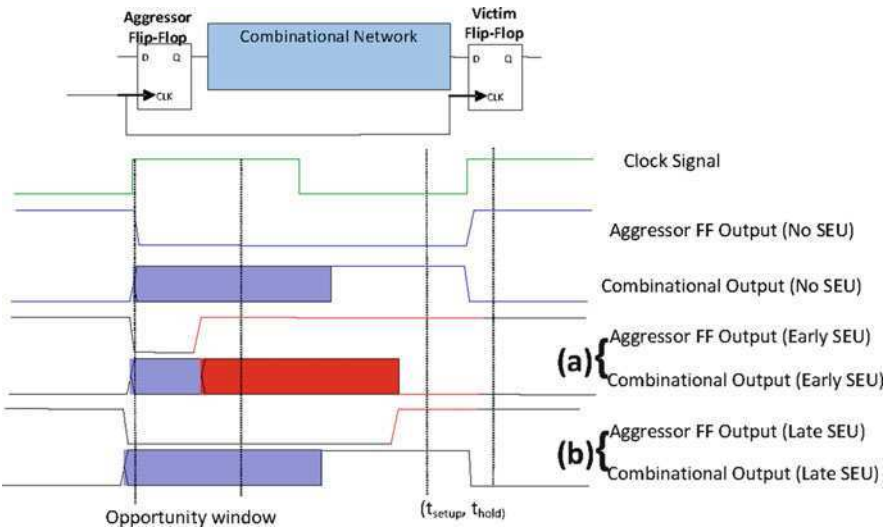


Fig. 5.8 Propagation of an SEU from the affected cell through the logic combinational network

opportunity window (Fig. 5.8b), then it will not have enough time to propagate through the logic network and to affect the destination flip-flops in time.

Particle strikes are random events totally uncorrelated with the clock period. Thus an SEU affecting a flip-flop can occur at any time in the clock period with a uniform distribution probability. The instant of its occurrence within the clock cycle together with the delay of the paths through which the induced error is propagated determines the instant of arrival of the error to the next stage flip-flops. This arrival instant determines the probability for the error to be captured by a flip-flop. Let t_{SEU} be the instant of occurrence of an SEU within the clock period $[0, T_{\mathrm{CLK}}]$ and *delay* be the delay of the activated path. Then, the probability that the induced error is captured by the flip-flop placed at the end of this path is given by

$$p_i = \begin{cases} 1 & 0 \le t_{\mathrm{SEU}} \le T_{\mathrm{CLK}} - t_{\mathrm{setup}} - \mathrm{delay} \\ \mathrm{Unknown} & T_{\mathrm{CLK}} - t_{\mathrm{setup}} - \mathrm{delay} < t_{\mathrm{SEU}} < T_{\mathrm{CLK}} + t_{\mathrm{hold}} - \mathrm{delay} \\ 0 & T_{\mathrm{CLK}} - \mathrm{delay} + t_{\mathrm{hold}} \le t_{\mathrm{SEU}} \end{cases} \tag{5.1}$$

For $0 \le t_{\mathrm{SEU}} \le T_{\mathrm{CLK}} - \mathrm{delay} - t_{\mathrm{setup}}$ the probability is 1, since in this case the erroneous value is present on the flip-flop input all along the setup and the hold time, while for $T_{\mathrm{CLK}} - \mathrm{delay} + t_{\mathrm{hold}} \le t_{\mathrm{SEU}}$, the probability is 0 because the correct value is present on the flip-flop input all along the setup and the hold time. In the other cases ($T_{\mathrm{CLK}} - t_{\mathrm{setup}} - \mathrm{delay} < t_{\mathrm{SEU}} < \mathrm{T_{CLK}} + \mathrm{t_{hold}} - \mathrm{delay}$), the erroneous value is present during a part of the setup and the hold time, and the correct value is present for the remaining part. In these cases, the behavior of the flip-flop is undetermined and the probability for latching the erroneous value

is set as unknown. However, this probability starts from the value 1 for $t_{SEU} = T_{CLK} - t_{setup} -$ delay (when the whole latching window is covered), and progressively decreases to 0 for $t_{SEU} = T_{CLK} + t_{hold} -$ delay (when there is no overlap between the transient pulse and the latching window). We can consider with a good accuracy that this decrease is linear along the time interval $[(T_{CLK} - t_{setup} -$ delay), $(T_{CLK} + t_{hold} -$ delay)].[2] In this case, the cumulative probability that an SEU occurring during this interval is latched by the next stage flip-flop will be equal to 0.5. Thanks to this assumption and from (1) we easily obtain the following value for the TDR factor:

$$\text{SEU\_TemporalDerating} = \int_0^{T_{CLK}} p_i = 1 - \frac{\text{delay} + \frac{t_{setup}}{2} - \frac{t_{hold}}{2}}{T_{CLK}} \tag{5.2}$$

We observe a notable difference between temporal derating (TDR) of master and slave stages from a flip-flop.

If the delay of the activated paths from the output of the affected flip-flop to the input of the following flip-flop is larger than the half-period of the clock cycle, only errors which occur quite early in the clock cycle can be captured in the following flip-flop. Thus, the upsets must occur in the master latch. Any upsets in the slave will have to be discarded, since they arrive too late to cause any damage.

If the activated paths are short, then some errors from the slave (when active) and all errors from the master (when active) will be latched in the downstream flip-flop.

Representing the temporal derating for single-event upsets (TDR-SEU) (Fig. 5.9), for given values of the delay parameter and of $t_{hold} - t_{setup}$, shows that the TDR-SEU is very important (the circuit is more sensitive) for lower working frequencies. When the circuit works close to the rated maximum speed, the slack of the considered activated path is very low, thus the opportunity window becomes very narrow. The flip-flop is sensitive during a short interval of time, thus diminishing the probability of the occurrence of an SEU at this short time interval. These results have to be considered carefully as they apply in operation under different clock frequencies of the same circuit. If a circuit is designed to operate at high speed (e.g., by increasing the number of pipe-line stages), the impact of the clock frequency will not be as significant as shown in Fig. 5.8, since in this case not only the clock period but the circuit delays will also be shorter.

[2]This means that the latching probability of a pulse will be given by $cw/(t_{setup} + t_{hold})$, where cw is the part of the latching window covered by the pulse. This assumption may not be accurate enough for individual values of cw. When $cw/(t_{setup} + t_{hold})$ is close to 0, the latching probability of the pulse will be lower than $cw/(t_{setup} + t_{hold})$, while for $cw/(t_{setup} + t_{hold})$ close to 1, the latching probability will be larger than $cw/(t_{setup} + t_{hold})$. However, when considering the cumulative probability for the interval $0 < cw/(t_{setup} + t_{hold}) < 1$, these differences will cancel one another at a large extent, resulting in a result of good accuracy.

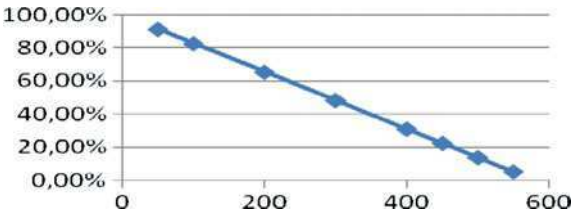


Fig. 5.9 TDR-SEU versus working frequency for a flip-flop (1.72-ns activated path)

5.3.3.2 Temporal Derating for Single-Event Transients

When the current pulse produced in a node of a sequential cell is strong enough to flip the state of a cell (creating an SEU), the output of the cell is permanently modified until the end of the clock cycle where a new value is latched. On the contrary, SETs affecting a combinational cell manifest as short pulses on the output of the affected cell. Thus, they can disappear before the end of the clock cycle. From the site of the occurrence of the SET, the transient pulse will propagate through the subsequent cells of the combinational logic until it reaches the inputs of some sequential cells (usually flip-flops). If the transient fault is memorized in the sequential cell, it will cause an error; otherwise, it will disappear without any effects. In order to be memorized, the fault must be present on the inputs of the sequential cells during their latching window. For a flip-flop, this window corresponds to the interval (t_{setup}, t_{hold}). The presence of the fault during this time interval is a function of its occurrence instant, of the delay of the propagation path, and of the width/waveform of the transient pulse at its arrival at the flip-flop input. This width/waveform depends on the initial SET PW/waveform and on its deformation as it propagates through the circuit cells. Since the arrival instant is random, we have used the notion of TDR for SETs in order to cope with this complexity [41].

Let's consider the case of a transient fault reaching the data input of a flip-flop (Fig. 5.10–5.12). Similar to the TDR-SEU, we will compute the probability that the error is latched in the flip-flop. For a given duration PW_{SET} of the transient pulse, this probability is determined by the instant of the arrival of the fault with respect to an opportunity window $[-t_{setup} - PW_{SET}; t_{hold}]$, defined around the latching edge t_{LE} of the clock (i.e., t_{LE} is considered as instant 0). In addition to the arrival time, the duration of the transient pulse is an important parameter as it will determine if the pulse could cover fully or only partially the latching window (t_{setup}, t_{hold}). Thus, if $PW_{SET} < t_{setup} + t_{hold}$, the transient pulse will never fully cover the latching window. In this case, according to its arrival time, the pulse will partially cover the latching window, giving an unknown latching probability; or it will not overlap with it, giving a latching probability equal to 0. On the contrary, if $PW_{SET} \geq t_{setup} + t_{hold}$, then, according to its arrival time, the pulse will fully cover the latching window, giving a latching probability equal to 1; it will partially cover the latching window, giving an unknown latching probability; or it will not at all overlap with it, giving a latching probability equal to 0. The detailed results are reported in (5.3).

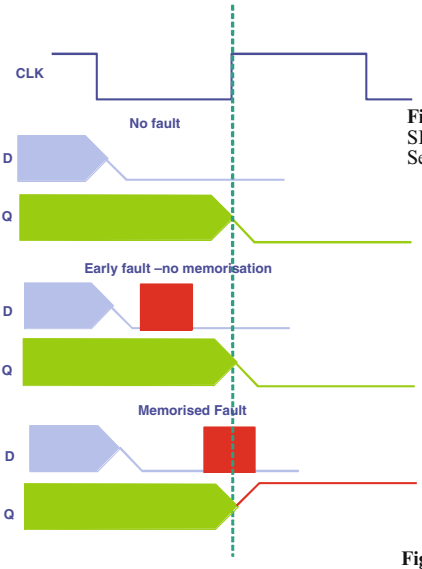


Fig. 5.10 SET on the data input of a D-FF

Fig. 5.11 SET on a Set/Reset cell

S	R	Fault on	Q-	Q+
0	0	S	-	1 (*)
0	0	R	-	0 (*)
0	1	S	0	x (*)
0	1	R	0	0
1	0	S	1	1
1	0	R	1	x (*)
1	1	S	x	0 (*)
1	1	R	x	1 (*)

* Incorrect behavior (error)

Fig. 5.12 SET on a D-latch

CLK	Fault on	Output
On	D	Fault propagation
On	CLK	No effect
Off	D	No effect
Off	CLK	Incorrect memorization

$$p_i(t) = \begin{cases} 1 & t_{\text{hold}} - \text{PW} \le t \le -t_{\text{setup}} \\ \text{Unknown} & -t_{\text{setup}} - \text{PW} \le t < t_{\text{hold}} - \text{PW};\ -t_{\text{setup}} < t \le t_{\text{hold}} \\ 0 & -t_{\text{setup}} - \text{PW} < t;\ t > t_{\text{hold}} \end{cases}$$
$$\text{If PW} \ge t_{\text{setup}} + t_{\text{hold}}$$

$$p_i(t) = \begin{cases} \text{Unknown} & -t_{\text{setup}} - \text{PW} \le t < t_{\text{hold}} \\ 0 & -t_{\text{setup}} - \text{PW} < t;\ t > t_{\text{hold}} \end{cases} \qquad \text{If PW} < t_{\text{setup}} + t_{\text{hold}} \tag{5.3}$$

As in the case of SEUs, we will consider that the latching probability of a pulse will be proportional to the fraction of the latching window it covers.[3] Thanks to this assumption, we can cope with the unknown probability values in (5.3). Then, we easily obtain the following value for the TDR factor:

$$\text{SET\_TemporalDerating} = \int_0^{T_{\text{CLK}}} p_i(t)\mathrm{d}t = \frac{\text{PW}}{T_{\text{CLK}}} \tag{5.4}$$

[3]For $\text{PW}_{\text{SET}} < t_{\text{setup}} + t_{\text{hold}}$, the cancellation of the variances from this assumption may not work as well as was discussed in the case of SEUs.

Inverter	INVx1	INVx2	INVx4
H\L delay	33ps	29ps	26ps
L/H delay	115ps	95ps	75ps

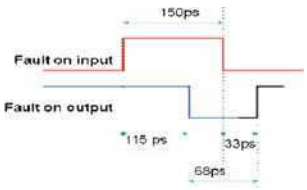


Fig. 5.13 SET propagation through a 130-nm inverter

In these formulas, we have used the PW term as the effective SET pulse width seen by the input of the flip-flop. However, the SET is deformed during the propagation through the logic network. The basic mechanisms that will modify the duration of the fault are various intrinsic cell delays, re-convergence of different paths on a common node, and progressive pulse shortening and filtering during the propagation through the logic paths.

The differences between the delays of a cell for different signal transition are able to lengthen or shorten the SET when the transient propagates through the cell (e.g. Fig. 5.13).

The re-convergent paths are very critical for the transient fault since the same fault may emerge in a single convergence node at several instants by propagating through different paths with various delays (Fig. 5.14). This way, the original SET becomes a burst of pulses with much larger total duration and, therefore, much higher probability to be memorized by some sequential cells.

Another phenomenon that affects the propagation of SETs consists in the progressive attenuation of the PW. Any SET shorter than the delay of a cell will be attenuated, or completely eliminated SETs which are not considerably larger than the intrinsic cell delays will propagate through few cells but they will be inevitably attenuated and then eliminated few stages later (Figs. 5.14 and 5.15). Only long enough SETs (a few times the cell delays) will survive through the propagation in the combinational network.

All these aspects should be considered during the TDR computation for SETs. It may require an important effort to be spent by the reliability engineer that wants to take in account the contribution of the SET to the global SER of the circuit. As a much more friendly method, we recommend using (5.4) as a simplified way to compute the SET TDR quickly.

By applying (5.4), we obtain the graphs from the Fig. 5.16 for SET TDR. The interpretation of these graphs shows that the TDR is higher (thus the circuit is more sensitive) for longer SET PWs (more energetic particles or a more aggressive working environment) or higher working frequencies (i.e., the SET becomes increasingly important with regard to ever-decreasing clock periods).

5.3.4 Logic Derating

Logic derating (LDR) consists in evaluating the propagation of the logic fault from the output of the affected cell to the inputs of a sequential/memory cell. According

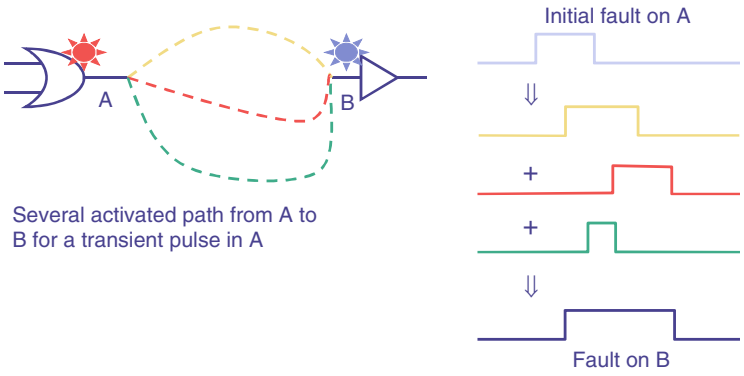


Fig. 5.14 Fault re-convergence and aggravation

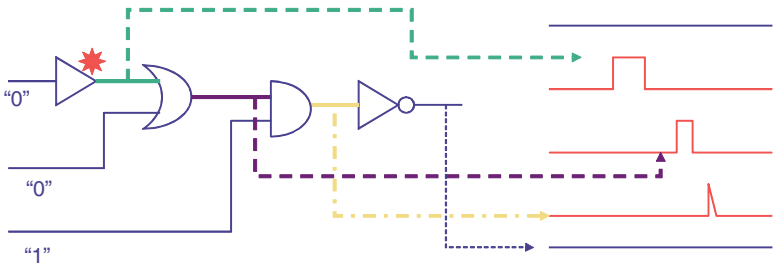


Fig. 5.15 Progressive SET attenuation

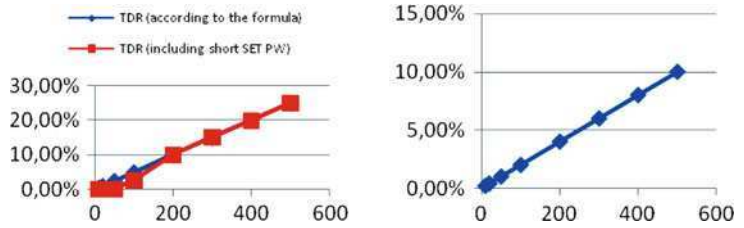


Fig. 5.16 TDR-SET evolution as a function of PW and working frequency. TDR versus SET pulse width (ps) for a 500-MHz working frequency. TDR versus working frequency (MHz) for a SET pulse width of 200 ps

to the state of the circuit (the values of the signals and cell outputs), the propagation of the fault is subject to logic blocking. As an example, an AND gate with a "low" input will block any faults on the other inputs.

The evaluation of the LDR is a common and well-researched subject in the reliability community [42–44]. Many research papers, methodologies, and tools are available to help the designer in evaluating the LDR of circuits. Standard simulators (*compiler-driven*, *table-driven*, or *event-driven*) are perfectly able to inject and simulate faults in very complex circuits and designs. Specific fault simulation

techniques such as *Parallel Fault Simulation, Single Pattern Parallel Fault Propagation, Parallel Pattern Single Fault Propagation*, and others have been devised for an optimized fault propagation evaluation [45–56]. Other techniques that do not require simulation, such as static LDR based on a probabilistic approach, are also available. In the Sect. 5.5, we will enumerate some of the actual techniques and tools that can be used for LDR.

5.3.5 Functional Derating

As mentioned earlier, the purpose of functional derating (FDR) is to evaluate whether the SE has any observable impact on the operation of the circuit, board, or system. It takes into account the actual usage of the circuit and the function of the system. Also, the criteria for deciding whether or not a failure has occurred should be set by someone knowledgeable about the architecture and the function of the system. Thus, FDR requires a strong collaboration between the reliability engineer (the SER expert) and the system engineers.

Additionally, the failure classification implies some degree of subjectivity, since a criticality parameter can also be added to the fault classes. Moreover, the usage of the circuit will have a strong impact on the failure analysis, since different modes/applications will exhibit different SE-related failures. If the considered circuit has a clearly stated function, then the FDR will have to consider this specific function and the effect of any relevant function-specific parameters. However, for general-purpose circuits such as CPUs, the possible application field is quite large, thus rendering the FDR analysis much more complex.

As stated, the FDR computation takes into account the propagation of the SE during several clock cycles. The goal is to evaluate whether (a) the SE has been silently discarded (dropped), without any further impact on the operation of the circuit, (b) it remains in the circuit in a latent state without any observable degradation of the operation (or function for instance in the case of FPGAs) of the circuit or worst, or (c) it produced a functional failure.

The evaluation of the functional impacts of faults and errors in microelectronic circuits and systems has also been extensively researched. In the following sub-chapter, we will present practical techniques and tools to help the reliability engineer in performing FDR analysis.

5.4 Dynamic SEE Analysis

5.4.1 Overview

This chapter presents simulation-based approaches for evaluating the behavior of the circuit in the presence of SEEs. The SEE simulation task can benefit from the rich existing know-how from the Design for Test (DFT) field. Fault simulation

techniques are widely researched and used in the electronic design practice. One of the most important challenges consists in dealing with the large number of faults to be taken into account. This is especially true for SETs that may arrive at any instant during the activity of the circuit and have widely different PWs.

The approach that is generally used for fault simulation is also useful for SEE simulation:

- First, the context is specified: the circuit representation and the fault model are defined according to the availability of circuit design files, validation environment, and simulation tools, as discussed earlier.
- Then, we proceed at the construction of the fault universe containing an enumeration of the faults that are considered for simulation. The circuit description is explored in order to find the circuit features that can be used with the appropriate fault models. A gate-level netlist will contain combinational cells that exhibit SETs on their outputs and sequential cells for which the SEU model is applied. A RTL description will produce a list of signals that will be later synthesized as flip-flops for which the SEU model is again applied. Large circuits will produce huge numbers of faults, thus some reduction (compaction) is required to diminish the number of faults to be simulated.
- Then, the simulation of the faults from the fault universe is performed, generally using some technique to accelerate the simulations. Additional techniques are used in order to determine if the simulated fault produces actual failures and to record the relevant error data. One important aspect of the simulation is the classification of the faults according to their effects on circuit functionality.
- Lastly, the simulation data are analyzed in order to present the user relevant data about the circuit behavior in the presence of the injected faults. Additional treatment of the fault data may be performed in order to extrapolate existing data for a more complete analysis report.

5.4.2 Gate-Level Netlist SEE Simulation

Simulating the SETs and SEUs using a gate-level netlist is the single method that generates the most accurate results for a given circuit implementation. Since we can use a detailed circuit representation containing per-cell structural information accompanied by timing data, the fault propagation through the circuit is representative of the real fault. The timing data are critical for SETs for which latching by a sequential element is a function of the PW, which is strongly affected during the propagation through the activated paths.

The first task is to explore the circuit netlist and build the fault universe. An exhaustive fault universe would contain faults on all the outputs of all cells for every clock cycle. Since SEEs may arrive at any instant in the clock period, we need to simulate faults whose instant of occurrence is uniformly distributed over the

clock period. This operation has the potential of greatly increasing the simulation time. Thus, the first improvement we propose is to simulate just one fault and use the results to determine the overall latching probability of faults for the uniform distribution of their occurrence instants. The single fault (an SET of a given width or an SEU) can be injected on a stable circuit, since the following can be shown [38]:

– From an SER perspective, the impact of an SET or SEU can be separately evaluated from the normal transitions activity of the circuit.
– The output waveform relevant for computing the overall fault latching probability does not depend on the instant of fault occurrence.

Thus, the simulation time is reduced drastically. Furthermore, the compaction of the fault universe can be performed using many of the existing techniques.

The fault injection is then performed using the fault models presented earlier and the simulator capabilities.

After the fault injection, we need to propagate the fault accurately through the circuit under evaluation. As the transient pulse is a very time-dependent phenomenon, it is very important to have precise and complete timing information on the circuit under evaluation. This information is also essential in order to determine the latching probability of SEUs (their chances to reach the next stage sequential elements during their latching window). Also, we need a simulator capable of taking advantage of the timing information and simulating the fault propagation. As we must inject faults modifying the logic value of a node during a specified time interval, we can use VHDL or verilog gate-level netlist representation of the circuit and a logic event-driven simulator. The netlist only offers structural information about the design. Timing information is usually provided by an SDF file. This is a very flexible and powerful method for adding delay information to the simulation. Also, we can take into consideration process variations and operating environments. The SDF file allows the specification of timings in the circuit in three scenarios: worst, nominal, or best. We can run the simulation for each scenario and compare the results.

The effects of SEUs and SETs on the circuit also depend on the values applied on the circuit inputs. Therefore, we must use a large set of input vectors and a large number of fault simulations for each vector to be able to compute the probability to latch a fault.

Accelerated simulation algorithms are available in order to keep the simulation time at acceptable levels [14, 15, 19]. As we use an event-driven simulator, the simulation time is proportional to the number of events evaluated by the simulator. So, we are interested in reducing the number of events using differential or serial fault simulation techniques. Important reduction is possible, thanks to the result mentioned earlier [14]. In fact, for each input vector, we simulate the circuit to determine the steady-state values of its nodes. Then, we inject in this steady state each fault. As a consequence, the events related to the normal input transitions are simulated only once. Then, for each fault, only the event induced by it is simulated.

The full-fledged fault simulation involving timing information is only required up to the end of the current clock cycle. This will determine whether the fault disappears or is latched and thus transformed into a soft error, whose simulation through the rest of the system can be performed at a higher level.

5.4.3 Behavioral/RTL/HLS SEE Simulation

A typical activity during the design flow is to describe the circuit using a higher level abstraction method such as RTL (in any of the widely used hardware description languages such as VHDL or Verilog). The designer works with the source of different blocks of the circuit and uses specific tools to write, build, simulate, and verify the design. During this design stage, most of the effort is spent in implementing the circuit as described in the functional specifications and performing the necessary validations. Also, the DV engineers will run a large series of simulations with various test cases in order to check the system function. The simulation environments used by the DV team is especially conceived to highlight any functioning errors and is especially optimized to minimize the simulation time in order to be able to run a large number of simulation campaigns. If any errors are detected during validation, the design engineers will then correct the problem and re-launch the validation campaign.

It is at this point that an SER analysis could be very beneficial since any SER-related effort could have a very important impact. SER-related problems could be solved during the improvement of the source RTL design; error-mitigation methods can be selected and/or designed and implemented; and any HW-SW error handling procedures could be devised [57–59].

Additionally, the DV simulation environment is a very appropriate solution for performing SER simulation, since the errors and failures induced by SEE faults will be analyzed by the DV procedures. Furthermore, since the DV tends to be exhaustive, the SEE impact on the available test cases/applications can be quickly analyzed.

There are, of course, a number of limitations. The first limitation concerns the available circuit representation. Most of the implementation effort is made using higher level source files (RTL). Thus, the implementation of the circuit in terms of gates is not yet known at this point. SET simulation is not possible and SEU analysis is hampered by the imprecision in associating register-like RTL signals to the exact flip-flops that will be used later. Thus, the RTL simulation will not be able to evaluate whether the SET/SEU fault became an SE, but rather if the SE may become a functional failure. In conclusion, this approach will compute the FDR factor.

A practical RTL simulation approach consists in building a fault universe comprising faults on the RTL signals that will be later synthesized as sequential cells. Then, several techniques are available for the compaction of this fault universe to a more manageable size.

Error injection can be performed using simulator capabilities (force/release, PLI/VPI procedures, etc.). Then, without any further intervention, the DV simulation environment should detect if the injected error had any impact on the functioning of the device. This approach is very simple to implement since it requires only an error injection procedure. The disadvantage is that no further effort to minimize the simulation time is performed, thus we require one full simulation run per error.

Fault simulation acceleration techniques are readily available and can be used to simulate several errors per run, or stop the simulation when the injected error has been classified as failure/latent/discarded. Additionally, with some, generally important, extra effort to implement the circuit in FPGA together with special fault injection and diagnosis circuitry, fast emulation methods allow to improve the simulation time drastically, and to enable emulating drastically larger numbers of faults (see also Chap. 6).

The outcomes of the RTL SEE simulation campaign are twofold:

- Quantitative results, such as the FDR factors for various cells/signals or blocks. These results will be useful during the final SER computation stage.
- Qualitative results about the behavior of the circuit in the presence of errors (classification of the different functional failures, evaluation of their severity, etc.) that will be used during the error-mitigation stage, especially when devising error protection strategies.

All these efforts are very consuming in terms of engineer time. Dedicated tools allow reducing this time substantially.

5.5 Static SEE Analysis

5.5.1 Overview

In the previous chapter, we have presented tools and methodologies for SEE simulation. These methods allow a precise characterization of the behavior of the circuit in a radiation environment. The time required for this analysis may be quite substantial and must be factored when scheduling the project. Additionally, the degree of interaction between reliability and design engineers is significant and limits the autonomy of the SER expert and also requires additional time for feedback to the design engineers for any SER-related improvement and optimization.

The design engineers often need reliable and fast tools to execute quick design iterations and incremental improvements of the circuit. This is especially true when performing specific analyses that require expert skills, such as SER evaluation. In most cases, the SER tools should be integrated in a standard design flow, such as fast prototyping platform, and use standard file formats and concepts. Additionally,

a great deal of SER data (such as the cell SER characterization data) should be made available to the various engineers during the design flow.

A solution to these needs could be offered in the form of SER tools that use static methods able to evaluate quickly the SER of the circuit or system. The derating factors are especially useful in this case, since the global SER can be expressed as a sum of the SER of the different parts of the circuit multiplied by various derating factors (5.5). This approach is also very flexible since the breakdown of the circuit in various parts can be made at any representation level. The gate-level netlist (5.6) allows for computing the contribution of each cell for the exact implementation of the circuit but requires detailed data. The block-level approach allows the system architect to evaluate and budget the SER of the whole system quickly in order to build an adequate error handling methodology and establish SER constraints and recommendations to the implementation team.

$$\mathrm{SER}_{\mathrm{System}} = \sum_{\mathrm{parts}} \mathrm{RAWSERpart} \cdot \mathrm{Derating} \tag{5.5}$$

$$\mathrm{SER}_{\mathrm{Gate-Level\ Netlist}} = \sum_{\mathrm{cell}} \mathrm{RAWSER}_{\mathrm{cell}} \cdot \mathrm{TDR} \cdot \mathrm{LDR} \cdot \mathrm{FDR} \tag{5.6}$$

5.5.2 Gate-Level

As discussed earlier, the gate-level netlist contains all the required structural information for accurately evaluating the contribution of standard cells and memory blocks. Each of these components will contribute to the global SER of the circuit following a formula similar to the (5.6). We will need to evaluate the TDR and LDR factors in order to assess the transformation of the SET/SEU faults into an SE. The FDR factor of the SE will be evaluated at a higher level.

5.5.2.1 Combinational Cells

The SER due to SETs in combinational cells can be computed using the (5.6). This requires the knowledge of the temporal derating for single-event transients (TDR-SET) and the LDR factors.

The TDR-SET could be evaluated using a simplified equation such as the simplified forms (5.4). To compute the effective SET PW at its arrival on the input of a downstream sequential cell, static timing analysis tools can be used. These tools are able to compute the deformation of the transient pulse during the propagation through the activated path. Additionally, upper and lower bounds can be set if required.

The LDR can also be evaluated using fast, mathematical methods [6]. These methods aim at computing a global probability of a fault to propagate through a logic path, for the whole test case or application, for all test vectors. The techniques

consist in exploring the whole circuit, and enumerating all the possible activated paths from the output of every cell to the inputs of sequential cells. Then, every activated path will be submitted to a mathematical evaluation that computes the probability of a fault to propagate through this path.

One technique [7, 8] consists in considering the probability of discrete standard cells to propagate faults on their inputs. As an example, inverters, buffers, and XOR-type cells will always propagate faults on any inputs. AND/OR gates will propagate faults on a specific input according to the state of the other inputs of the cell. Thus, we will need to know the "1" or "0" probabilities of the states of every cell outputs or signals of the circuit. The state distribution can be computed using several methods that have been proposed for testing purposes and especially for works related to the observability of the circuit [9, 10]. Also, a fairly simple technique of good accuracy, though not as good as a probabilistic analysis since it does not allow considering conditional probabilities, consists in monitoring the signal waveforms during a fault-free simulation and counting the occurrences of "1" and "0."

5.5.2.2 Sequential Cells

The techniques that we have used for combinational cells could also be deployed successfully for SEUs in sequential cells. Particularly, the computation of the LDR factor is the same for both types of cells.

The most important difference consists in the computation of the TDR. The formulas used in this chapter require the knowledge of the slack on various activated paths between two successive stages of flip-flops. This slack can be very easily computed using standard synthesis or timing analysis tools. These tools are able to use a very precise description of the circuit, optionally annotated with parasitics information, in order to evaluate the timing parameters of the circuit in a short amount of time.

Again, lower and upper bounds can be set for the computed TDR. From an SER perspective, the worst case (highest TDR, circuit most sensitive) can be found by considering the largest possible opportunity window. This is obtained by using the shortest register-to-register slack to determine this window.

5.5.3 Behavioral/RTL

As discussed earlier, the RTL circuit representation only provides accurate information about the memorization elements of the circuit, sequential, or memory cells. Thus, no implementation-specific SER analysis can be performed, particularly the SET contribution cannot be considered correctly.

Thus, the only possible computation is related to the contribution of SE in sequential cells and in memory blocks. The SEUs in sequential cells must first be transformed to SE by using the appropriate derating factors. However, computing the TDR-SEU is not yet possible, since circuit timing data are not yet available. Thus, the reliability engineer should evaluate a preliminary TDR-SEU based on previous experience, design synthesis rules and practices, initial performance estimation, and other factors.

Additionally, the exact type of the actual flip-flop cells that will be used in the final implementation netlist cannot be known at this point. Since the intrinsic sensitivity of various flip-flops may be different, the SER expert should choose a representative flip-flop or a representative intrinsic SER value in order to use this value for computing the SER of a sequential cell as the product of the intrinsic SER, by the preliminary TDR-SEU, $\mathrm{LDR}_{\mathrm{STAGE}(i+1)}$, and $\mathrm{FDR}_{\mathrm{STAGE}(i+1)}$. That is $\mathrm{SER}_{\mathrm{CELL}} = \mathrm{SER}_{\mathrm{CELL\_INTR}} \times \mathrm{TDR\text{-}SEU} \times \mathrm{LDR}_{\mathrm{STAGE}(i+1)} \times \mathrm{FDR}_{\mathrm{STAGE}(i+1)}$. This is because an SEU affecting a sequential element of stage (stage i) is transformed into a SE after propagation through the downstream logic until reaching the next stage of sequential elements (stage $i + 1$). Thus, we apply the TDR-SEU and the $\mathrm{LDR}_{\mathrm{STAGE}(i+1)}$, corresponding to this propagation, in order to transform the SEU into a SE. Then, we apply the functional derating $\mathrm{FDR}_{\mathrm{STAGE}(i+1)}$ of stage $i + 1$. The FDR of stage $i + 1$ can be determined by multiplying the FDR of stage i by the LDR of stage $i + 1$. Thus, we obtain the simpler expression $\mathrm{SER}_{\mathrm{CELL}} = \mathrm{SER}_{\mathrm{CELL\_INTR}} \times \mathrm{TDR\text{-}SEU} \times \mathrm{FDR}_{\mathrm{STAGE}(i)}$, where the LDR factor is eliminated (absorbed by $\mathrm{FDR}_{\mathrm{STAGE}(i)}$).

The memory blocks' contribution to the SER can be computed as a product of the capacity of the memory by the intrinsic SER (per megabit). This process is facilitated by the fact that at this point in the design flow, the list of memory blocks, the type, and even implementation details (the internal organization/mux factor) have been already decided upon by the system architect. Thus, we should have a fairly accurate perspective of the nominal SER of all the memory blocks. However, this nominal value should be adjusted by the effect (efficiency) of any error-mitigation techniques, if implemented. Then, the FDR factor should also be evaluated in order to take into account the actual usage of the memory for the given application.

Thus, the last item on our list concerns the evaluation of the FDR factor. We have seen earlier that we can compute this factor by using fault simulation techniques. However, the object of this chapter is to present alternative methods that could be quickly used instead of simulation. We should clearly state from the beginning that these alternative methods are intrinsically less precise than fault simulation.

One practical approach is to use a "Utilization Derating (UDR)" factor. This parameter is nominally a part of the more-encompassing FDR concept and can be computed using an estimation of how much the specific circuit block is used for a given application. As an example, a memory block has a nominal capacity. However, in specific cases, not all the memory is used, thus the percentage of actual used memory should be taken into account when computing the memory SER.

Another technique that can be applied for memories is the access profile, meaning the list of write and read operations in the memory. This information could be extracted from a single fault-free simulation, or provided by the designer of the system. This list can be exploited in order to compute the lifetime of the data stored in the memory (the time interval between a write and the latest read before the next write). The ratio of the total lifetime for each memory word versus the total simulation time can be used to further derate the memory SEU.

The UDR factor is also useful for standard cells blocks, in the case where the designer is able to provide some estimation of the duty factor of specific blocks. As an example, the FPU block of a CPU may be not used at all for an industrial control application, thus the designer's contribution to the SER could be ignored or minimized. However, for a mathematically intensive application, the FPU could be one of the most intensively used parts of the circuit.

In addition, preliminary FDR factors can be estimated by the SER engineer using expertise acquired from previous projects or manual estimations about the inner workings of the circuit.

5.5.4 Architecture/Block

This circuit representation is the poorest in terms of actual circuit structure. As the SEEs are strongly dependent on the type of cells and the topology of the circuit, the summary information available for a block-level circuit is not enough to perform an accurate SER analysis. However, since SEEs are a part of the global reliability of the system, the design flow will surely benefit from an early analysis of the overall sensitivity of the system. The architect will be able to decide whether the computed SER is reasonable with regard to reliability constraints that are set by the end user or the application. Furthermore, SER-required decisions can be made at an early stage in the design flow, with a minimal impact on the implementation effort.

A practical block-level SER analysis approach consists in enumerating all the blocks of the design that can be reasonably foreseen at this point. Then, quantitative structural information is estimated for each block. The information should consist in a very preliminary quantity of standard cells, memory blocks, IP blocks, and other features. Intrinsic raw SER is associated with each type of feature, and derating factors are then evaluated. The derating factors could be computed based on previous SER experience, reliability and design engineers feedback, data from similar circuits, etc. Then, the overall sensitivity of the system is evaluated using the (5.5).

An example of such flow is illustrated in the Fig. 5.17.

The results of this analysis should allow the system architect to check the computed SER against initial specifications and to take relevant and informed actions in order to improve the reliability of the design.

Block	Feature	Size	Raw SER	TDR	LDR	FDR	ECC	MBU%	SER
CPU.ALU	Seq	100000	800	40%	80%	90%	No		22
	Comb	500000	100	10%	40%	80%	No		2
CPU.CTRL	Seq	25000	800	90%	70%	90%	No		11
	Comb	100000	100	10%	40%	80%	No		0
CPU.L1	SRAM	32768	800	100%	100%	100%	No		25
CPU.L2	SRAM	2097152	600	100%	100%	80%	Yes	3%	24
SystemRAM	SRAM	16777216	1000	100%	100%	25%	Yes	1%	40
SystemROM	FLASH	33554432	1	100%	100%	50%	No		16
IOInterface	Seq	1000000	300	80%	80%	20%	No		37
	Comb	5000000	50	5%	40%	80%	No		4
Buffer	DPSRAM	65536	1000	100%	100%	25%	No		16
								Total	196

Fig. 5.17 An example of an architecture SER analysis

In the case where the preliminary estimations shows a risk to exceed the SER specifications, the architect could decide on an error-mitigation strategy by recommending the implementation engineers to use error handling mechanisms on critical parts of the circuit. SRAM memories are prime candidates for ECC or parity protection. The error-mitigation recommendations could target individual components (memory instances) or whole blocks.

Another action is to choose a different standard cell library or founder (if feasible) containing cells with improved SER characteristics. Else, some critical blocks could be implemented using lower sensitivity flip-flops (or even hardened flip-flops), again if available and if other considerations (power dissipation, area, and timing) are also met. At some point, tradeoffs between reliability, area, speed, and power may be necessary.

Lastly, the system architect, together with the implementation engineers, both hardware and software, should elaborate a common strategy and goals for handling errors. According to the circuit function, correcting an error has a certain downside (either hardware – silicon overhead, or performance decrease when correcting) [60]. With respect to the application, different types of errors have specific costs (downtime) and impact (ranging from none to show-stopper). Each error can be optimally handled with specific techniques (hardware or software); thus the collaboration of all the actors involved in the design of the system is required.

The following table presents a synthesis of the possible abstraction levels and the elements that must be evaluated during SER analysis:

Abstraction level	Feature type	Fault	Fault to error derating	Error	Error to failure derating	Failure
Netlist	Combinational cell	SET	LDR TDR	SE	FDR	Functional failure
	Sequential cells	SEU	LDR TDR	SE		
	Memory instances	→		SE		
RTL/behavior	Sequential cells	SEU	TDR	SE		
	Memory instances	→		SE	Estimated FDR	
Architecture	Standard cell blocks	SET, SEU	Estimated LDR, TDR	SE		
	Memory instances	→		SE		

5.6 Conclusion

The various SER computation approaches, at cell and macro-block level, netlist, RTL, and architecture levels, based on dynamic or static techniques, using various derating factors, probabilistic or simulation approaches, are very consuming in terms of engineer time. Given the complexity of today's' circuits and the difficulty of accurate SER analysis, the reliability engineer will greatly benefit from dedicated SEE EDA tools. Also, selecting the most adequate analysis approach and making the mitigation decisions at the most adequate abstraction level may have an important impact on the design time and the SER mitigation cost. Finally, having access to accurate SER data for standard cells, memories, and other IPs, coming from radiation test results or from dedicated cell and IP level, SER estimation tools are another burden for the designers.

Furthermore, SEE constraints can be added very early in the system architecture to better disseminate and propagate SER constraints to all the actors of the design flow. In addition, the overall system reliability will be improved by optimizing system resilience to SEEs at the same time as "classic" metrics such as speed, area, timing, power consumption, and yield. Adequate EDA tools and methodologies can help tremendously in minimizing the SEEs' impact, avoiding unnecessary design iterations, and reducing time-to-market.

We hope that this chapter offers a comprehensive presentation of the risks, solutions and possible benefits when correctly addressing the SER evaluation challenge and will be useful for people from academia interested in establishing new classes or starting a new research in the SER domain as well as for engineers having to deal with increased SER constraints in their designs.

References

1. "SEU Induced Errors Observed in Microprocessor Systems". Asenek, V., et al. 1998, IEEE Transactions on Nuclear Science, Vol. 45, No. 6, p. 2876.
2. "Satellite Anomalies from Galactic Cosmic Rays". Binder, D., Smith, E.C. and Holman, A.B. 1975, IEEE Transactions on Nuclear Science, Vols. NS-22, No. 6, pp. 2675–2680.

3. “Single Event Upsets in Implantable Cardioverter Defibrillators”. Bradley, P.D. and Normand, E. 1998, IEEE Transactions on Nuclear Science, Vol. 45, No. 6, p. 2929.
4. “Circuit Simulations of SEU and SET Disruptions by Means of an Empirical Model Built Thanks to a Set of 3D Mixed-Mode Device Simulation Responses”. Belhaddad, H., et al. 2006, RADECS2006.
5. “A New Approach for the Prediction of the Neutron-Induced SEU Rate”. Vial, C., et al. 1998, IEEE Transactions on Nuclear Science, Vol. 45, No. 6, p. 2915.
6. “Simulation Technologies for Cosmic Ray Neutron-Induced Soft Errors: Models and Simulation Systems”. Tosaka, Y., et al. 1999, IEEE Transactions on Nuclear Science, Vol. 46, No. 3, p. 774.
7. “Soft Error Modeling and Mitigation” Tahoori, M. 2005, EMC Presentation.
8. “Production and Propagation of Single-Event Transient in High-Speed Digital Logic ICs”. Dodd, P.E., et al. 2004, IEEE Transactions on Nuclear Science, Vol. 51, No. 6, pp. 3278–3284.
9. “Single Event Transient Pulsewidth Measurements Using a Variable Temporal Latch Technique”. Eaton, P., et al. 2004, IEEE Transactions on Nuclear Science, Vol. 51, No. 6, pp. 3365–3369.
10. “Single Event Transients in Deep Submicron CMOS”. Hass, K.J. and Gambles, J.W. 1999, IEEE 42nd Midwest Symposium on Circuits and Systems, pp. 122–125.
11. “Single Event Transient Pulsewidths in Digital Microcircuits”. Gadlage, M.J., et al. 2004, IEEE Transactions on Nuclear Science, Vol. 51, No. 6, pp. 3285–3290.
12. “Measurement of Single Event Transient Pulse Width Induced by Ionizing Radiations in CMOS Combinational Logic”. Perez, R., et al. 2006, RADECS Proceedings.
13. “Contribution a la definition des specifications d’un outil d’aide a la conception automatique de systemes electroniques integres nanometriques robuste”. Doctoral Thesis, Perez, R. 2004.
14. “Measuring the Width of Transient Pulses Induced by Ionising Radiation”. Nicolaidis, M. and Perez, R. 2003, Proceedings of the International Reliability Physics Symposium, p. 56.
15. “Modeling and Verification of Single Event Transients in Deep Submicron Technologies”. Gadlage, M.J. Schrimpf, R.D. Benedetto, J.M. Eaton, P.H. Turflinger, T.L., Proceedings of IEEE International Reliability Physics Symposium Proceedings, 2004, pp. 673–674
16. “Modeling of Transients Caused by a Laser Attack on Smart Cards”, Leroy, D., Piestrak S.J., Monteiro F. and Dandache, A., Proc. IOLTS’05 – 12th IEEE Int. On-Line Testing Symposium, Saint Raphael, France, July 6–8, 2005.
17. “Synthetic Soft Error Rate Simulation Considering Neutron-Induced Single Event Transient from Transistor to LSI-Chip Level”. Hane, M., Nakamura, H., Uemura H., et al. 2008, Proc. of SISPAD, pp. 365–368.
18. “Study on Influence of Device Structure Dimensions and Profiles on Charge Collection Current Causing SET Pulse Leading to Soft Errors in Logic Circuits”. Tanaka, K., Nakamura, H., Uemura T., et al. 2009, Proc. of SISPAD.
19. “Single Event Effects in Static and Dynamic Registers in a 0.25 μm CMOS Technology”. Faccio, F., et al. 1999, IEEE Transactions on Nuclear Science, Vol. 46, No. 6, pp. 1434–1439.
20. “SEU Testing of a Novel Hardened Register Implemented Using Standard CMOS Technology”. Monnier, T., et al. 1999, IEEE Transactions on Nuclear Science, Vol. 46, No. 6, p. 1440.
21. “Detailed Analysis of Secondary Ions’ Effect for the Calculation of Neutron-Induced SER in SRAMs”, Hubert, G., et al. 2001, IEEE Transactions on Nuclear Science, Vol. 48, No. 6.
22. “SEU Response of an Entire SRAM Cell Simulated as one Contiguos Three Dimensional Device Domain”. Roche, Ph., et al. 1998, IEEE Transactions on Nuclear Science, Vol. 45, No. 6, p. 2534.
23. “Determination of Key Parameters for SEU Occurrence using 3-D Full Cell SRAM Simulations”. Roche, Ph., et al. 1999, IEEE Transactions on Nuclear Science, Vol. 46, No. 6, p. 1354.
24. “An Alpha Immune and Ultra Low Neutron SER High Density SRAM”. Roche, Ph., et al. 2004, Reliability Physics Symposium Proceedings, 42nd Annual, pp. 671–672.
25. “Process Impact on SRAM alpha-Particle SEU Performance”. Xu, Y.Z., et al. 2004, Proceedings of IRPS.

26. “Alpha-Particle-Induced Soft Errors In Dynamic Memories”. May, T.C. and Woods, M.H. 1979, IEEE Transactions on Electron Devices, Vols. ED-26, No. 1, p. 39487.
27. “Neutron Soft Error Rate Measurements in a 90-nm CMOS Process and Scaling Trends in SRAM from 0.25-μm to 90-nm Generation”. Hazucha, P., et al. 2003, Proceedings of IEDM.
28. “Soft Error Rate Increase for New Generations of SRAMs”. Granlund, T., Granbom, B. and Olsson, N. 2003, IEEE Transactions on Nuclear Science, Vol. 50, No. 6, pp. 2065–2069.
29. “Neutron-Induced SEU in Bulk and SOI SRAMs in Terrestrial Environment”. Baggio, J., et al. 2004, Reliability Physics Symposium Proceedings, Vol. 42nd Annual, pp. 677–678.
30. “Low-Energy Neutron Sensitivity of Recent Generation SRAMs”. Armani, J.M., Simon, G. and Poirot, P. 2004, IEEE Transactions on Nuclear Science, Vol. 51, No. 5, pp. 2811–2816.
31. “Single Event Effects in Avionics”. Normand, E., IEEE Transactions on Nuclear Science, 1996, Vol. 43, Issue 2, Part 1, pp. 461–474.
32. “Single Event Upset at Ground Level”. Normand, E., IEEE Transactions on Nuclear Science, 1996, Vol. 43, Issue 6, Part 1, pp. 2742–2750.
33. “SRAM SER in 90, 130 and 180 nm Bulk and SOI Technologies”. Cannon, Ethan H., et al. 2004, Proc. 42nd Int’l Reliability Physics Symp, p. 300.
34. “Characterization of Multi-Bit Soft Error Events in Advanced SRAMs”. Maiz, J. Hareland, S. Zhang, K. Armstrong, P., Proceedings of IEEE International Electron Devices Meeting 2003, pp. 21.4.1–21.4.4.
35. “Neutron Induced Single-word Multiple-bit Upset in SRAM”. Johansson, K., et al. 1999, IEEE Transactions on Nuclear Science, Vol. 46, No. 6, p. 1427.
36. “Analysis of Local and Global Transient Effects in a CMOS SRAM”. Gardic, F., et al. 1996, IEEE Transactions on Nuclear Science, Vols. Vol. 43, No. 3, p. 899.
37. “Large System Soft Error Rate (SER) Qualification”. Eagan, D.J., et al. 1992, Proceedings of Custom Integrated Circuits Conference, pp. 18.2.1–18.2.4.
38. “Simulating Single Event Transients in VDSM ICs for Ground Level Radiation”. Alexandrescu, D., Anghel, L. and Nicolaidis, M. 2004, Journal of Electronic Testing: Theory and Applications.
39. “Accurate and Efficient Analysis of Single Event Transients in VLSI Circuits”. Reorda, M.S. and Violante, M. 2003, Proceedings of the 9th IEEE International On-Line Testing Symposium.
40. “A Systematic Approach to SER Estimation and Solutions”. Nguyen, H.T. and Yagil, Y. 2003, Proceedings of the International Reliability Physics Symposium, pp. 60–70.
41. “Probabilistic Estimates of Upset Caused by Single Event Transients”. Hass, K.J. 1999, 8th NASA Symposium on VLSI Design, pp. 4.3.1–4.3.9.
42. “An Accurate SER Estimation Method Based on Propagation Probability”. Asadi, G. and Tahoori, M.B. 2005, Proceedings of Design, Automation and Test in Europe Conference.
43. “On Testability Analysis of Combinational Networks”. Brglez, F. 1984, Proceedings of IEEE Symposium on Circuits and Systems, pp. 221–225.
44. “A Model for Transient Fault Propagation in Combinatorial Logic”. Omana, M., et al. 2003, Proceedings of the 9th IEEE International On-Line Testing Symposium.
45. “High Performance Parallel Fault Simulation”. Varshney, A.K., et al. 2001, Proc. Intl. Conf. on Computer Design: VLSI in Computers & Processor (ICCD 01), Vols. 1063–6404/01.
46. “Data Parallel Fault Simulation”. Amin, M.B. and Vinnakota, B. 1995, Proc. Intl. Conf. on Computer Design: VLSI in Computers & Processor (ICCD 95), Vols. 1063-6404/95, pp. 610–616.
47. “Static Analysis of SEU Effects on Software Applications”. Benso, A., et al. 2002, Proceedings of the International Test Conference, pp. 500–508.
48. “HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits”. Lee, H. and Ha, D.S. 1996, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 15, No 9, pp. 1048–1058.
49. “An Efficient, Forward Fault Simulation Algorithm Based on the Parallel Pattern Single Fault Propagation”. Lee, H. and Ha, D.S. 1991, Proc. the IEEE Intl. Test Conf. on Test: Faster Better, Sooner, pp. 946–955.

50. “Fault List Compaction Through Static Timing Analysis for Efficient Fault Injection Experiments”. Reorda, M.S. and Violante, M. 2002, IEEE Symp. on Defect and Fault Tolerancein VLSI Systems, pp. 263–271.
51. “Speeding-up Fault Injection Campaigns in VHDL models”. Parrotta, B., et al. 2000, 19th International Conference on Computer Safety, Reliability and Security, Safecomp, pp. 27–36.
52. “Emulation-based Fault Injection in Circuits with Embedded Memories”. García-Valderas, M., et al. 2006, Proceedings of IEEE International on Line Testing Symposium, Como Lake, Italy.
53. “An Extension of Transient Fault Emulation Techniques to Circuits with Embedded Memories”. García-Valderas, M., et al. 2006, DDECS.
54. “Fast Timing Simulation of Transient Fault in Digital Circuits”. Dharchoudhury, A. 1994, Proc. Intl. Conf. on Computer-Aided Design, pp. 719–726.
55. “Analog-Digital Simulation of Transient-Induced Logic Errors and Upset Susceptibility of an Advanced Control System”. Carreno, V., Choi, G. and Iyer, R.K. 1990, NASA Technical Memo.
56. “New Techniques for Speeding-up Fault-Injection Campaigns”. Berrojo, L. Gonzalez, I. Corno, F. Reorda, M.S. Squillero, G. Entrena, L. Lopez, C., Proceedings of IEEE Design, Automation and Test in Europe 2002, pp. 847–852.
57. “Design for Soft Error Resiliency in Internet Core Routers”, Silburt, A.L., Evans, A., Burghelea A., Wen, S.-J. and Alexandrescu, D., IEEE Transactions on Nuclear Science.
58. “A Multi-Partner Soft Error Rate Analysis of an InfiniBand Host Channel Adapter”, Chapman, H., Landman, E., Margarit-Illovich, A., Fang, Y.P., Oates, A.S., Alexandrescu, D. and Lauzeral, O., SELSE 2010.
59. “Complex Electronic Systems Soft Error Rate (SER) Management”, Alexandrescu, D., Wen, S.-J. and Nicolaidis, M., Special Session on Soft Errors in Electronic Systems, ETS 2009, http://www.iroctech.com/pdf/ALE_ETS_2009_Presentation.pdf.
60. “Evaluation of Soft Error Tolerance Technique Based on Time and/or Space Redundancy”. Anghel, L., Alexandrescu, D. and Nicolaidis, M. 2000, Proc. of 13th Symposium on Integrated Circuits and Systems Design, pp. 237–342.

Further Reading

61. “Radiation Effects on Microelectronics in Space”. Srour, J.R. and McGarrity, J.M. 1988, Proceedings of the IEEE, Vol. 76, No. 11, p. 1443.
62. “Modeling the Cosmic-Ray-Induced Soft-Error Rate in Integrated Circuits: An Overview”. Srinivasan, G.R. 1996, IBM Journal of Research and Development, Vol. 40, No. 1, p. 77.
63. “The Design of Radiation-Hardened ICs for Space: A Compendium of Approaches”. Kerns, S.E. and Shafer, B.D. 1988, Proceedings of the IEEE, Vol. 76, No. 11, p. 1470.
64. “Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic”. Shivakumar, P., et al. 2002, Proceedings of the International Conference on Dependable Systems and Networks, p. 389–398.
65. “Contribution of Device Simulation to SER Understanding”. Palau, J.-M., et al. 2003, Proceedings of the International Reliability Physics Symposium, pp. 184–189.

Chapter 6
Hardware Fault Injection

Luis Entrena, Celia López-Ongil, Mario García-Valderas, Marta Portela-García, and Michael Nicolaidis

Hardware fault injection is the widely accepted approach to evaluate the behavior of a circuit in the presence of faults. Thus, it plays a key role in the design of robust circuits. This chapter presents a comprehensive review of hardware fault injection techniques, including physical and logical approaches. The implementation of effective fault injection systems is also analyzed. Particular emphasis is made on the recently developed emulation-based techniques, which can provide large flexibility along with unprecedented levels of performance. These capabilities provide a way to tackle reliability evaluation of complex circuits.

6.1 Introduction

As technology progresses into nanometric scale, the concern for reliability is growing. The introduction of new materials, processes, and novel devices along with increasing complexity, power, performance, and die size affect reliability negatively. On the contrary, the reduction in dimensions, capacitance, and voltage results in less node critical charge, bringing up the soft-error threat. Actually, taking into account all these trends, the soft-error rate (SER) per bit is expected to keep stable, according to recent studies [1]. However, since the memory bit count and the functionality integrated in logic components are increasing rapidly, the threat of soft errors is becoming a reality for many applications where it was not a concern in the past. The increasing use of electronic systems in safety critical applications,

L. Entrena, C. López-Ongil (✉), M. García-Valderas, and M. Portela-García
Electronic Technology Department, Carlos III University of Madrid, Spain
e-mail: entrena@ing.uc3m.es; celia@ing.uc3m.es; mgvalder@ing.uc3m.es; mportela@ing.uc3m.es
M. Nicolaidis
TIMA (CNRS, Grenoble INP, UJF), Grenoble–France,
e-mail: michael.nicolaidis@imag.fr

M. Nicolaidis (ed.), *Soft Errors in Modern Electronic Systems,*
Frontiers in Electronic Testing 41, DOI 10.1007/978-1-4419-6993-4_6,
© Springer Science+Business Media, LLC 2011

where human life is at stake, forces to ensure dependability and makes it an important challenge today.

Providing quality of service in the presence of faults is the purpose of fault tolerance. But before a fault-tolerant system is deployed, it must be tested and validated. Thus, dependability evaluation plays an important role in the design of fault-tolerant circuits. Fault injection, i.e., the deliberate injection of faults into a circuit under test, is the widely accepted approach to evaluate fault tolerance. Fault injection is intended to provide information about circuit reliability covering three main goals: validate the circuit under test with respect to reliability requirements; detect weak areas that require fault-tolerance enhancements; and forecast the expected circuit behavior in the occurrence of faults.

From a general point of view, we can distinguish between hardware and software fault injection, although the frontier between them is not well defined. Software fault injection deals with software reliability and will not be treated here. Hardware fault injection is related to hardware faults, which are generally modeled at lower levels (e.g., logical or electrical) and are injected into a piece of hardware.

In spite of the work made over many years, hardware fault injection is still a challenging area. New types of faults and effects come to place or achieve increasing relevance. In addition to permanent stuck-at faults or transient faults affecting memory bits, such as single-event upsets (SEUs), today designers must face the possibility of timing faults, single-event transients (SETs) affecting combinational logic, and multiple bit upsets (MBUs) affecting memories. More complex circuits need to be evaluated as a consequence of technology scaling and increasing density. In particular, Systems on Chip (SoCs) include a variety of components, such as microprocessors, memories, and peripherals, which pose different fault injection requirements. The widespread use of field-programmable technology confronts the need to evaluate the effect of errors on the configuration bits. As complexity increases, the number of faults to be injected in order to achieve statistical significance also increases. Thus, there is a need for new approaches and solutions in order to accurately reproduce fault effects, increase fault injection performance, and support the variety of existing technologies and components.

This chapter summarizes the current state of the art in hardware fault injection techniques and optimizations of the hardware fault injection process. It must be noted that the fault injection process is not only concerned with the means to inject faults. A complete environment is required for initialization of the circuit under test, selection and application of appropriate workloads, collection of information about faulty circuit behavior, comparison with the correct behavior, classification of fault effects, and monitoring of the overall process. The importance of each of these tasks must not be neglected, because all of them are relevant for a successful evaluation.

The remaining of the chapter is organized as follows. Section 6.2 reviews the most relevant hardware fault injection techniques and the existing approaches to inject faults. Section 6.3 describes the fault injection environments. Section 6.4 describes optimizations that contribute to increase fault injection performance. Finally, Sect. 6.5 contains the conclusion of this chapter.

6.2 Hardware Fault Injection Techniques

Dependability evaluation of modern VLSI circuits entails the need of injecting realistic faults at internal locations, and observing in an efficient way the circuit behavior in the presence of these faults. Indeed, with the generalization of deep submicron technologies, there can be a much greater number of faults in digital systems, and a significant proportion of them occur internally in the chip [2]. There are different methods for injecting faults in integrated circuits mimicking hard or soft errors. Some of these methods have been used for several years, while others are being proposed in recent research works.

In every fault injection method, there are some common elements that may be defined at different abstractions levels. First of all, faults to be injected in the evaluation process should be selected. The possibility of provoking real faults within the device or just modeling the effect they cause (fault model) in the circuit elements must be considered. Second, the circuit to be checked, usually named device under test (DUT) or circuit under test (CUT), could be a commercial-off-the-shelf (COTS), a prototype, or a design model. The level of abstraction for the DUT is related to the type of fault to be injected. Third, a collection of workloads should be available in order to get a representative subset of the circuit functionality. Circuit robustness should be checked when working as closely as possible to normal operation in its final application. Finally, the most important element in every fault injection method is the expected result, which corresponds mainly to a measure of device robustness against faults.

Furthermore, any fault injection method requires a specific setup for the fault injection campaign. In particular, a dedicated PCB with the DUT must be developed. Also, a system for workload application and processing of the test results, including hardware, software, communication links, etc., should be implemented for results processing.

Faults are typically classified according to their duration into permanent, intermittent or transient faults. Permanent faults are related to manufacturing defects and circuit aging. Transient faults are mainly caused by the environment, such as cosmic radiation or electrical noise. They do not produce a permanent damage and their effects are known as soft errors.

Cosmic radiation is the main source of single-event effects (SEEs) in integrated circuits. SEEs are caused by single energetic particles and take many forms, with permanent or transient effects. Thus, in this case, injected faults range from real faults coming from natural cosmic radiation to fault models of SEEs; circuits to be tested range from COTS to design descriptions. The result of a fault injection campaign is the probability a device will fail when working. Typically, this measure is the failure in time (FIT), which stands for the number of circuit failures per 10^9 h, and it is referred to a given radiation environment.

The main hardware fault injection cases will be summarized in the following subsections. Some of them are also addressed in detail in other chapters.

6.2.1 Physical Fault Injection

Physical fault injection methods use external perturbation sources, such as natural and accelerated particle radiation, laser beam, pin forcing, etc. The objective of this test is the analysis of circuit robustness in the presence of faults affecting a device. These methods are applied on COTS or prototypes for qualifying new technologies or existing chips for a new application environment.

These methods can cause a wide range of internal damages in the circuit under test: SEEs, displacement damage (DD) and total ionization dose (TID). Typically, studied SEEs are SEUs, SETs and single-event latchups (SELs).

6.2.1.1 Radiation Methods

The most traditional method for provoking internal soft errors in the circuit under test is the use of particle radiation. Cosmic radiation is the main source of SEEs in integrated circuits. Therefore, testing a device in its real environment (space, high altitude, etc.) is the most realistic way of evaluating its sensitivity with respect to SEEs. There are some practical disadvantages for this solution that are related to cost and time-to-market. Due to the low probability of error, weeks or months are generally required, as well as hundreds or thousands of samples, for obtaining valid measures. Another disadvantage is the unknown relationship between failures and the energy of particles striking the samples.

When shorter testing times and more controlled experiment setup are required, *accelerated radiation tests* are used for qualifying new technologies. The DUT receives a beam of particles, coming from a specific accelerator facility [3] or from a radioactive source [2]. In this case, few samples are needed (typically in the order of ten) as well as less time for testing (hours or days). Also, an energy or intensity sweep can be applied on the particle beam affecting the circuit under test. In this case, it is easy to know the SER with respect to the energy of particles.

In accelerated radiation tests, several types of particles are used for evaluating circuit robustness in harsh environments. Particles coming from cosmic radiation (primary or secondary radiation) are heavy ions, protons, and neutrons. Also, shells of devices emit alpha particles that provoke SEEs in the circuit.

The main origin of cosmic radiation is the sun [4]; it provokes ionizing particles (heavy ions and protons), known as primary radiation, in deep space and stratospheric orbits, and non-ionizing particles (neutrons), known as secondary radiation, in atmospheric applications. The IEEE standard for testing space-borne components [5] indicates the type of particles present in different environments. This information is used together with the final location of the circuit for stating the work environment and deciding the type of radiation test to be performed. When checking the dependability of circuits in aircrafts or in earth surface, neutron and alpha particles are used for accelerated radiation tests [6]. Alpha particles affect circuits, especially at earth surface. On the contrary, heavy

ions are used for testing circuits working in nuclear or spatial applications [7]. Finally, protons are employed for testing components of terrestrial satellites [8, 9].

Standards developed by space agencies [8, 9] or by JEDEC association [6, 7] regulate radiation test procedures. In any case, simulation software tools are used together with these tests in order to obtain a more accurate knowledge of damage effect in devices. Material, type of particle, orbit, etc., are key elements in these calculations.

In these experiments, the setup task is a heavy process. Flux, fluence, and energy of particles must be set accurately (dosimetry is a key factor) for achieving a significant number of events and avoiding TID damage. The final result obtained is the cross-section, which is a function of particle energy or linear energy transfer (LET) and gives the number of events detected with respect to the particle fluence applied.

Static and dynamic tests should be performed on the DUT. While static tests qualify the technology of a device, dynamic tests measure the robustness of a circuit running in that device with a given workload. It is possible to disaggregate both tests and avoid dynamic test under radiation. Velazco et al. [10] proved that dynamic test results can be obtained by combining static test results (static cross-section) and results obtained with another fault injection method.

The main disadvantages of fault injection based on accelerated radiation ground testing are the high cost of the test campaigns and the relative small number of events achieved per run, which may lead to results that are not statistically significant. Also, controllability and observability are very limited. In any case, this type of test is currently mandatory for qualifying a technology in aerospace applications.

6.2.1.2 Laser Methods

A relatively recent approach for injecting faults from an external source is the use of laser beams. Laser incidence in the internal elements of the circuit causes effects similar to the ones provoked by particles issued from cosmic radiation. Indeed, this method is associated with bit-flip fault model for SEU effects. It is able to inject faults in a very accurate way, with the help of a microscope and a laser beam spot control.

There are research works [11–13] that prove the correlation between the results obtained from accelerated radiation test and laser test. Although there is a slight difference between the particle–material interaction and the photon–material interaction, SERs obtained from laser beam exposure are commonly accepted nowadays [13].

Laser test provides a high level of accessibility to locate the circuit elements where faults are injected. Also, this method implies less expensive equipment than radiation ground test facilities and less complex experiment setup (e.g., it is not necessary to separate the DUT in another PCB).

6.2.1.3 Pin Forcing

Another solution for fault injection from external sources is pin forcing [14, 15]. It was proposed for testing relatively simple ICs. Some authors considered that forcing values at input/output pins of a device could provoke the same effect as SEEs in very simple circuits. There are several CAD tools developed for helping designers to execute fault injection campaigns, such as RIFLE [16], SCIFI [17], FIST [18], or Messaline[19]. Considering current complexities in ICs, this method is very limited. Currently, it is employed for testing other external aspects of reliability (vibrations, electrical noise, etc.), but it is not intended for SEU fault injection.

Although it is a really cheap solution, possible circuit damage due to values forced onto device pins, together with the poor controllability and observability provided in the increasingly complex ICs, makes this method unattractive for dependability analysis of current technologies.

Physical fault injection methods provide realistic measures of SERs, but they are very expensive. They are considered the best methods available for qualifying new technologies. Nevertheless, better solutions are required for testing circuits during the design process where re-design is possible and cheaper. Furthermore, very complex designs require testing large amounts of faults in order to obtain statistically valid results. When thinking of large fault sets for current designs, fault injection can be accomplished at higher abstraction levels.

6.2.2 Logical Fault Injection

Logical fault injection methods use logic resources of the circuit to access internal elements and insert the effect a fault provokes (fault model). These extra logic resources are originally intended for other purposes, such as the IEEE standard for Boundary Scan 1149.1 (JTAG) that provides an easy way for accessing internal scan path chains through a serial interface. Also, some commercial microprocessors include on-chip debugging (OCD) capabilities that enable access to internal memory elements (program counter, user registers, etc.). Finally, reconfiguration resources for programmable devices enable to control and observe internal configuration nodes and, therefore, injecting faults and observing their effects.

The undertaken fault models depend on the robustness analysis under execution. Therefore, bit-flip model is applied for SEUs and MBUs, stuck-at model is applied for permanent faults, voltage pulses are used for SETs, etc.

6.2.2.1 Software Implemented Fault Injection

SWIFI is intended for testing hardware by means of executing specific software that modifies internal memory elements (user accessible) according to a fault model.

Fault injection can be performed during compilation time or during execution time [20]. In this last group, typical approaches use timers, such as FERRARI tool [21], or interruption routines, such as XCEPTION tool [22] or CEU tool [10]. More recently, new solutions have been presented [23] combining software-based techniques with previous approaches.

6.2.2.2 On-Chip Debugging for Microprocessors

Debugging resources provide direct access to internal registers, program counter, and other key elements in microprocessor architectures. This access can enable fault injection and fault effects observation in a rapid and effective way. Furthermore, the external accessibility of these capabilities makes the automation of fault injection campaigns easier. The use of OCD resources for testing purposes has been studied by some authors in recent years. FIMBUL tool uses JTAG interface for injecting bit-flip faults into memory elements of a microprocessor [17]. Rebaudengo et al. use the Motorola OCD, named background debugging mode (BDM), to execute fault injection through a serial port [24]. Also, NEXUS debugging standard is being used to enhance this fault injection method [25]. In [26] and [27], solutions are proposed by implementing specific hardware modules for interfacing between DUT (microprocessor) and host machine.

Recently, Portela et al. [28] have proposed another enhancement in the use of OCD capabilities, implementing in a hardware module the host in charge of injecting faults and analyzing obtained results. By reducing communication delays between hardware and software, fault injection process can be easily accelerated and automated.

6.2.2.3 Reconfiguration Resources

Reconfiguration resources in programmable devices make possible a direct fault injection within memory elements in prototyped designs. This method is widely used to evaluate the effect of faults in the configuration memory of FPGAs (field programmable gate arrays), which is a very important issue in these devices.

Partial reconfiguration reduces the time needed for performing fault injection in the configuration memory of FPGAs. Ref. [29] presents a solution based on reconfiguring by means of the Xilinx software JBits. In [30], a tool for injecting SEU faults in a Virtex® FPGA is proposed. This tool is able to inject faults in programmable interconnections, which are not accessible through commercial software tools (JBits). In recent contributions, Alderighi et al. proposed the FLIPPER fault injection platform which enables the fault-tolerance evaluation of hardened prototypes in FPGAs [31].

6.2.3 Logical Fault Injection by Circuit Emulation

As FPGA-based prototyping becomes popular for ASIC verification, it can also be exploited for hardware fault injection. In this case, the circuit under test is prototyped in one or several FPGAs. This approach is generally known as emulation-based fault injection. Contrary to the approaches mentioned in the previous section, FPGAs are used here just as a means to support fault injection, and the final circuit is to be implemented in some ASIC technology.

Fault injection in an FPGA-based prototype can take advantage from the flexibility of field-programmable hardware. Fault injection requires high controllability of each circuit node in order to modify its logic state. This can be obtained by using the FPGA reconfiguration mechanisms to modify the circuit or the contents of accessible memory elements. Another approach consists in inserting some additional hardware blocks in the prototype to support fault injection. These hardware blocks are called *instruments*.

Emulation-based fault injection was originally developed for permanent faults. In [32], a fault injection method is proposed for stuck-at faults. This method consists in modifying the circuit by connecting a signal to a constant logic value. Therefore, the FPGA must be resynthesized and reconfigured for each fault. Several techniques to emulate faults in parallel are proposed to alleviate the resynthesis and reconfiguration effort.

A fault injection technique for SEUs (bit-flips) based on run-time reconfiguration of the FPGA is proposed in [33]. In this technique, flip-flop (FF) contents are modified by controlling the asynchronous set/reset of each FF through the FPGA configuration bitstream. Injection of a fault is performed with the following steps: (a) at injection time, read the states of FFs; (b) reconfigure FPGA to set the asynchronous set/reset switch of each FF as to keep the current state, except for the faulty FF, that will be set in the opposite way; (c) pulse global set/reset line (state of faulty FF is modified); and (d) reconfigure FPGA again to set the asynchronous set/reset switches to the original value. The first step is performed by readback of the configuration bitstream, which includes the states of FFs. Readback is also used to check the results of each fault injection experiment and classify fault effects. This idea is also followed up in the FT_UNSHADES platform [34].

Note that this approach requires reconfiguring twice the FPGA for each fault. Although partial reconfiguration can be used, the reconfiguration process is slow. Fault injection rates range between 0.1 s and more than 1 s per fault, depending on the length of the partial reconfiguration bitstream.

Circuit instrumentation is a means to overcome the limitations of FPGA reconfiguration. It consists in inserting some pieces of hardware or instruments that can provide external controllability and observability to inject a fault and observe its effects. Then, the circuit is prototyped in an FPGA including the instruments. It is important that circuit instrumentation can be automated in order to avoid handling the circuit and to make the instrumentation process effective. On the contrary, the

instruments should be small enough to obtain an acceptable overhead in the prototype.

In one of the earlier works [35], Hong et al. proposed a technique to avoid reconfiguring for each fault by adding some specialized hardware blocks. Each block is attached to a target node and contains a flip-flop that stores the injection signal value. These flip-flops are arranged in a chain, so that faults can be injected by shifting in the desired injection values in the chain.

A circuit instrumentation technique for the injection of non-permanent faults is proposed in [36]. This technique is intended to emulate SEUs by injecting faults in the circuit flip-flops. For this purpose, the circuit under test is modified by substituting each flip-flop by the instrument shown in Fig. 6.1. This instrument contains an additional flip-flop, called the mask flip-flop, and two gates that implement the mask logic. The mask flip-flop is used to select the fault injection target. At injection time, the inject signal is asserted to all instruments. Then, the input value at the nodes where the mask flip-flop is set will flip and the fault is injected. More elaborated instruments have been proposed by Lopez et al. [37] that will be described in Sect. 6.4.

The mask flip-flops are arranged in a scan chain that can be loaded serially. Once the mask scan chain is loaded, the inject signal is asserted at the required time to inject a fault. Faults can be injected at different nodes by shifting in the mask scan chain. MBUs are supported by setting more than 1 bit in the mask. Eventually, the contents of the functional flip-flops can be loaded into the mask flip-flops. This operation captures the internal state of the circuit, which can be observed by shifting out the mask chain.

The circuit instrumentation technique is very efficient, as it does not require reconfiguring the circuit for every fault. Also, setting the fault injection mask is much faster than FPGA reconfiguration. In addition, latent faults can be detected by checking the circuit state as obtained through the scan chain after the fault injection process. Experimental results [36] show a fault injection rate of 10,000 faults/s by

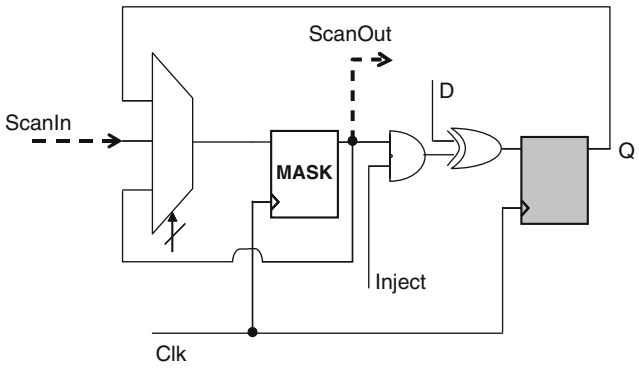


Fig. 6.1 Fault injection instrument in [36]

using a 20-MHz clock and a short workload (100 test vectors). For large workloads, the fault injection rate is inversely proportional to the workload length.

Injecting and propagating SETs is much more difficult, since it requires prototyping the logic delays of the circuit under test. Synthesizing the circuit under test for an FPGA would produce an equivalent functional circuit model, but with different gate delays. Existing approaches for SET emulation are based on embedding timing information in some way, such as the topology of the circuit [38]. Recently, an efficient approach has been proposed using a quantized delay model [39]. In this model, gate delays are rounded to a multiple of a small amount of time, addressed as time quantum. Quantized delays can be implemented in an FPGA using shift registers where the time quantum corresponds to a clock cycle. Experimental results show a fault injection rate in excess of one million faults/s, representing an improvement of three orders of magnitude with respect to a simulation-based approach.

The flexibility provided by FPGAs can be used to support some fault injection functions. For example, the FPGA prototype can include two instances of the circuit under test that are dedicated, respectively, to prototype the golden (fault-free) and the faulty circuit. Both instances run in parallel and the outputs can be compared inside the FPGA at the end of the execution to detect failures [34, 40]. A refined solution consists in duplicating just the sequential elements, sharing the combinational logic [37]. In this case, the golden and faulty instances run in alternate clock cycles.

Most of the work on emulation-based hardware fault injection focuses on general-purpose logic, but circuits may also include embedded memories. As the number of storage elements in memories is generally very large, memories are in fact very relevant. However, the controllability and observability are limited to a memory access per memory port and clock cycle.

Memories can be emulated by forcing the synthesizer to treat them as flip-flops. This approach would usually produce emulation circuits much larger than commercially available FPGAs. Therefore, embedded memories must be instrumented in a particular way to support fault injection.

A memory can be implemented in an FPGA using FPGA memory blocks. In this case, fault injection is performed by instrumenting the memory buses [28]. In [40], memories are implemented using dual-port FPGA memory blocks, where one of the ports is specifically devoted to fault injection. At the fault injection time, the target memory position is read, XORed with the fault mask, and then written back. A monitor circuit is included to clear all the internal memory after an experiment, in order to avoid accumulation of errors, and to read serially all the internal memory in order to compare it and detect faults. These two operations take a lot of time, just in proportion to memory size, and must be performed for every fault injection experiment. Thus, injecting a fault in a memory position can be achieved by instrumenting the memory buses, but the initialization of the memory and the extraction and comparison of the fault injection results for analysis are the major problems of memory fault injection.

6.3 Fault Injection System

A hardware fault injection system is able to execute a circuit with a workload in the presence of faults, and compare the faulty behavior with the fault-free behavior. A fault injection system is typically composed of the following elements:

- The CUT
- A fault injection mechanism, which can be physical or logical
- A test environment, in charge of the following tasks:
  - Supply the vectors required for the workload
  - Check the effect of faults in the CUT
  - Collect results
  - Classify faults
  - Control the whole process

Fault injection systems are used to perform fault injection campaigns, which are experiments intended for obtaining a measurement of the circuit reliability. Some examples of this measurement are the fault dictionary, the circuit cross-section, the SER, the mean time between failures, etc.

Fault injection systems using physical fault injection require a prototype of the DUT, which is exposed to the fault provoking element (radiation, laser). The system has to be built in such a way that the DUT is correctly exposed, but the rest of the system is not affected by the fault source. In accelerated radiation experiments, the CUT must be separated from the rest of the system, in order to receive the beam without affecting the test environment. For example, the THESIC system [41] consists of a motherboard as test environment, and a mezzanine board for the CUT. For laser campaign, the circuit package must be removed for the laser beam to be effective, and the CUT must also be visible through a microscope to locate the laser incidence point [42].

Systems using logical fault injection have a similar structure but use a different fault injection mechanism, so there are no restrictions related to physical exposure aspects. The only additional part to add is a fault injection method, for example, a host-controlled JTAG interface connected to the OCD of the CUT.

The rest of this section will cover the implementation of systems using logical fault injection by circuit emulation. This kind of system uses FPGA-based prototyping to implement the CUT. As this is the most general and flexible scenario, there is a large variety of solutions. Building these systems represent a very challenging task as far as the process performance is concerned.

There is a wide range of possibilities to build emulation-based fault injection systems, as there are many tasks that can be executed in a host computer or in the hardware. Fig. 6.2 shows the main required tasks. The user interaction takes place at the host computer and the circuit emulation is performed in the hardware (circuit core). The rest of the tasks may be executed either by the host computer or by the hardware.

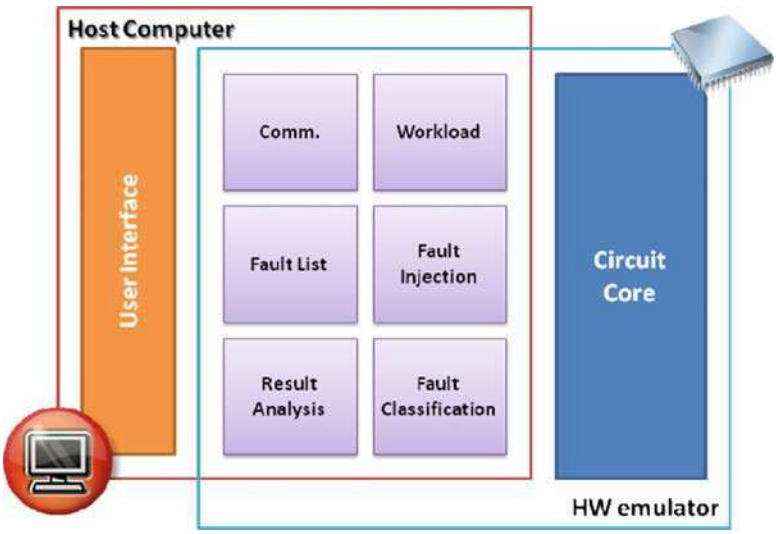


Fig. 6.2 Emulation-based fault injection system components

There are a lot of intermediate possibilities, depending on the tasks assigned to the host computer and to the hardware. The speed of the communication channel is critical when considering the amount of information to transfer.

In general, the tasks required to perform a fault injection campaign comprise fault list management, workload application, fault injection, fault classification and result analysis. These tasks are analyzed in the following paragraphs.

6.3.1 Workload

A workload must be provided to the circuit under evaluation for execution. The implementation of this task must consider a trade-off between flexibility, performance, and resource usage. Several approaches can be considered. Test vectors can be generated at the host computer and sent to the hardware when they are going to be applied. This method is very flexible, as the workload can be changed very easily, but implies a continuous host–hardware communication which slows down the execution.

On the contrary, a stimulus generation block can be implemented in the hardware, next to the circuit core, so it can supply vectors at the speed they are required. A simple approach is to use some BIST-like vector generation circuit, like an LFSR. This kind of implementation is very fast and uses very little resources, but the workload obtained may not be very representative.

An intermediate solution is to store test vectors in the FPGA memory. This solution is flexible, as the workload can be easily changed by downloading a new

one or by reconfiguring the FPGA, and it is also fast, because test vectors are fed to the circuit core by hardware. However, FPGA devices usually have a limited amount of internal memory, representing a drawback for this method. In order to improve resource usage, test vectors may be compressed, or external memory may be used if it is available on the FPGA board.

6.3.2 Fault List

The fault list management task has some important aspects, both in design and implementation. The ideal case is to generate a fault list including faults at every location and every time instant, for a given workload. Considering bit-flip fault model for SEUs, the complete single fault list would include a fault in every memory element and every clock cycle of the workload. This approach is practical only if the circuit is either very small or the fault injection system is very efficient, like Autonomous Emulation [37].

Usually, the system is not so efficient to perform a fault injection campaign with all possible faults in a reasonable time. In these cases, the fault space must be sampled to obtain a statistically representative subset. There are several approaches to create the fault list. The simplest one is to use random fault list generation, both in fault localization and time instant, although it must be taken into account that a computer or a hardware generated list is not really random, but pseudo-random. Other proposals include the use of Poisson distribution for the generation of the time instants, in order to reproduce the results of a radiation experiment. A deeper discussion on these aspects can be found in [43].

Regarding implementation, the fault list can be generated at the host computer or in the hardware. If it is generated at the host computer, it must then be transferred to the hardware. It is advisable to implement an intermediate storage mechanism, so that the emulator does not need to wait for the new fault when it has already finished processing the previous one. This mechanism can make use of internal FPGA memory or on-board memory.

In this case, the impact in performance is not as high as for the workload case. For the workload, a test vector is required every clock cycle, but a new fault is required only when the previous one has already been processed.

6.3.3 Fault Classification

The classification of a fault can be made out of the comparison of the faulty and the golden executions.

If a fault is injected and after some time it produces a result different than expected, it is called a *failure*. If the fault effect completely disappears from the circuit after some execution time, the fault is called *silent*. If the faulty circuit shows

differences with the golden one after the execution of the workload, but it did not produce any error in the results, the fault is called *latent*. If the circuit has some built-in fault detection mechanism, the faults can also be classified as *detected* or *not detected*. In microprocessor-based circuits, a fault could also be classified as *lost of sequence*, when the effect of the fault is modifying the normal instruction sequence, preventing the circuit to reach the end of the workload.

Concerning failures, the condition of producing an erroneous result is different for every case. For a control circuit, an error can be a value in the outputs that is different than expected. For an algorithm processor, the calculation results must be checked and they can be written at the circuit outputs or stored in memory. Failures could also be sub-classified by a criticality-based criteria. For example, some errors could produce physical damage in the system (e.g., an electronically controlled mechanical engine) or produce a dangerous situation (a brake system), while others may be irrelevant (a wrong pixel in an image).

Latent faults represent an additional problem. The system must have a mechanism to compare the golden and the faulty circuit states to decide if the fault is still present. For example, two instances of the circuit can be implemented, and a mechanism to compare them must be included. If partial reconfiguration is used, readback of the circuit flip-flops can be used for this purpose with a high penalty on performance [34]. With instrumented circuit technique, flip-flops are duplicated and compared [37], so that latent faults can be detected online.

Performance will profit from an early fault classification mechanism. Being able to stop the execution immediately after the fault is classified will allow saving some time. If the classification is made in hardware, it is easier to implement a fast classification mechanism. This aspect will be explained in more detail in the next section.

6.3.4 Result Analysis

The results of a fault injection campaign can be expressed in several ways. In terms of circuit qualification, it is usually required to obtain a single figure for the circuit reliability, like the SER, expressed by either number of FIT, or Mean Time Between Failures (MTBF). These figures are calculated using information from fault injection campaigns and taking into account the environment where the circuit will operate. Emulation-based fault injection campaigns are useful to obtain information about the consequences of faults. Information about fault occurrence probabilities and other aspects must be obtained using other methods.

The most complete result information that can be obtained from a fault injection campaign is the fault dictionary, which is the list that holds the classification of every injected fault. The complete fault dictionary is very useful to locate weak areas in the circuit or critical tasks in the workload.

The implementation problems for this task are quite similar to those of the fault list. A new result is generated for every processed fault. Results can be transferred

immediately after processing to the host, or they can be temporally stored in the hardware to improve performance.

In case result generation is very fast or there is no temporal storage available, statistical measures may be collected in the hardware. For example, results can be classified per location, or per injection instant, or just percentages can be calculated.

6.3.5 Communication

Emulator–host communication has a great impact on the system performance. In order to improve the performance, we can either increase the speed of the communication channel or decrease the amount or the frequency of the transmitted data.

Commonly used communication mechanisms are serial ports, USB, Ethernet, or PCI.

Obviously, an increment in the channel speed will result in an overall speed improvement, but several aspects must be taken into account. For example, communication channels like USB can be very fast, but only in burst mode, transmitting big amounts of data in a single pack.

In the case of a fault injection system, the information to transmit (test vectors, fault list, fault dictionary, and control commands) can be sparse in time. This approach is not very efficient for high speed communication channels, so it is advisable to design the emulator, including data compaction or communication buffers. In order to obtain the maximum performance, the objective is to maintain the core emulation circuit running as much time as possible, and avoid the time gaps due to communication.

6.4 Fault Injection Optimizations

Emulation-based fault injection techniques have the capability of notably speeding up the fault-tolerance evaluation process regarding other methods. Injecting millions of faults in a few hours is possible using these techniques. However, reducing this time is a very interesting goal, since fault-tolerance evaluation is a task performed many times during the circuit development. Furthermore, current circuits have large complexity, including an increasing count of sensitive areas that can be affected by faults. Therefore, the higher the number of possible faults, the higher the number of faults that must be injected to obtain a significant measurement of the circuit robustness. Definitely, speeding up the fault-tolerance evaluation process is required.

Several approaches exist to speed up the fault injection process using emulation-based techniques. In the following sections, we will describe the main sources of fault injection inefficiency and solutions to overcome them.

6.4.1 Autonomous Emulation

Emulation-based techniques profit from the capability of an FPGA to emulate circuit behavior at hardware speed. Using typical emulation-based fault injection solutions, the emulation process is interrupted every time the emulator needs to wait for the host to apply the stimuli, to inject a fault, or to check the output values. Then, a very intensive interaction is required between the emulator and the host computer. The host controls the injection and evaluation of every fault. This introduces a performance bottleneck due to the communication between the emulator and the host computer, which prevents taking full advantage of the FPGA capabilities for fast hardware emulation.

Autonomous emulation [37] is a fault injection solution aimed at avoiding the intensive communication between host and emulation platform. It consists in implementing the whole injection system in the FPGA by making use of the instrumentation mechanism to insert faults in the circuit under test (Fig. 6.4). The FPGA is in charge of performing the following tasks:

1. Managing the whole fault injection process.
2. Applying the input stimuli to the circuit under test.
3. Activating the fault injection.
4. Watching the circuit behavior under faults and classifying the injected fault depending on its effect on the circuit functionality.

Using autonomous emulation, access to any circuit-sensitive element is simple and straightforward, and the required time necessary to perform the different injection tasks can be significantly reduced. Figure 6.3 shows the typical emulation-based solutions scheme and Fig. 6.4 the autonomous emulation scheme with the purpose of illustrating the differences between both systems. The autonomous emulation system benefits from the available resources in current FPGA platforms, like memory blocks, in order to implement more tasks close to the circuit under test, which minimizes the required interaction with the host computer. The enhancements provided by the autonomous emulation solution are as follows:

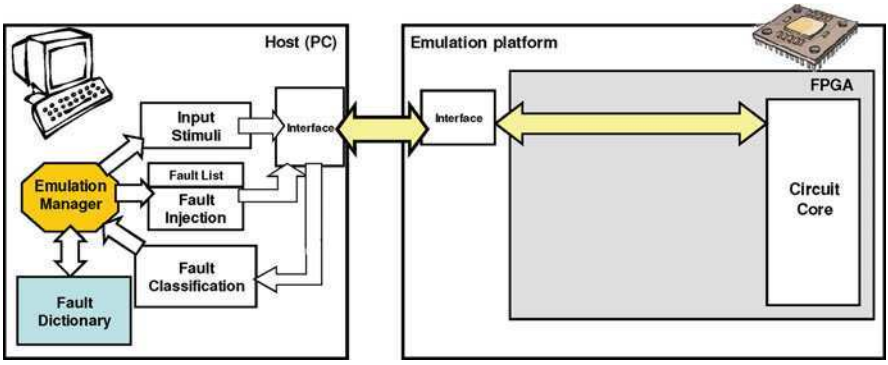


Fig. 6.3 Typical scheme for an emulation-based fault injection system

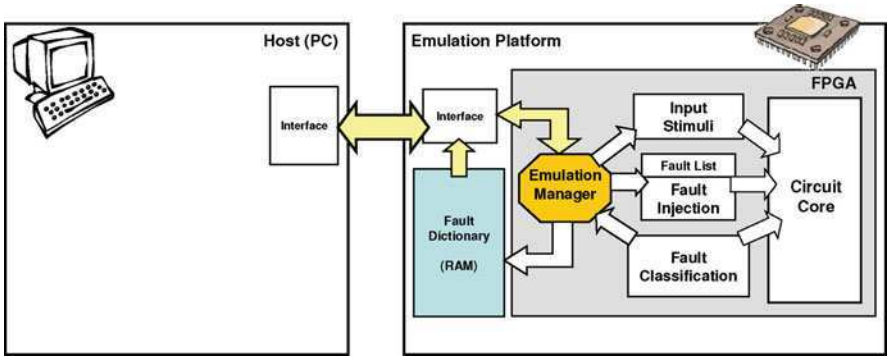


Fig. 6.4 Autonomous Emulation scheme

- The required communication between host computer and the emulation platform is minimized, being established only twice, at the beginning of the evaluation process to configure the FPGA from the PC, and at the end of the fault injection campaign to collect the obtained results, that is, to download the fault dictionary.
- Observability and controllability are significantly enhanced, since access to the memory elements does not require a particular communication channel and then it is straightforward and easier. In general, the typical emulation-based techniques set a trade-off between the process speed and the observability of the internal circuit resources, because higher observability requires more information exchange and, therefore, more time to spend in the evaluation process. The autonomous emulation system provides a high observability without penalty in the injection process speed, since the injection system and the circuit under test are implemented in the same device.
- Hardware implementation makes the parallel execution of different injection tasks and it speeds up the whole process with respect to a software implementation.

Once the PC-FPGA communication, that is, the main limitation in fault emulation techniques, has been minimized, the fault injection process is much more efficient. Moreover, the access to the circuit internal resources does not require exchange of information between the host computer and the FPGA. This feature allows the application of new optimizations to reduce the time spent per fault.

6.4.2 Fault Evaluation Process

The fault injection process can be optimized by applying techniques to reduce the time spent in the different steps needed to evaluate the consequences of a fault. For a given workload these steps are the following (Fig. 6.5):

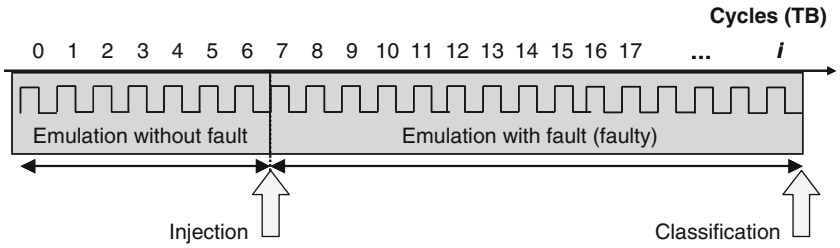

Cycles (TB)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 ... i
Emulation without fault
Emulation with fault (faulty)
Injection
Classification

Fig. 6.5 Time spent to emulate each fault

1. Reach the circuit state corresponding to the injection instant. The most common way to do it consists of running the workload from the beginning until the injection instant.
2. Inject the fault.
3. Classify the fault according to its effect on the circuit behavior. For this purpose, the circuit under test resumes the workload execution since the injection instant until the fault is classified or the workload finishes.

In the worst case, emulating the complete workload for each fault can be required. A fault injection campaign with a large number of injected faults and long workloads may involve excessive time to complete the evaluation process. With the instrumentation-based mechanism, the fault injection task takes just one clock cycle and so possible time optimization should be applied to the other steps (1 and 3). The time required for reaching the circuit state at the injection instant can be optimized by applying techniques to save fault-free emulation time. A solution consists in doing a previous storage of the circuit state and a posterior reloading of this state in the next fault injection (state restoration). Regarding fault classification, techniques can be applied to speed up fault emulation by aborting execution as soon as the fault can be classified, profiting from the higher observability available in an Autonomous Emulation system. In the following sections, these optimizations are detailed for SEU faults.

6.4.3 State Restoration

The circuit under test can get to the state corresponding to the injection instant in two different ways:

- Emulating the workload until the injection instant is reached.
- Storing the required state in memory elements of the circuit and restoring it just before the fault injection instant (state restoration). In this case, additional hardware to store the corresponding state is necessary.

State restoration avoids the fault-free circuit emulation for every injected fault. Required states are easily obtained from the golden execution, run just once.

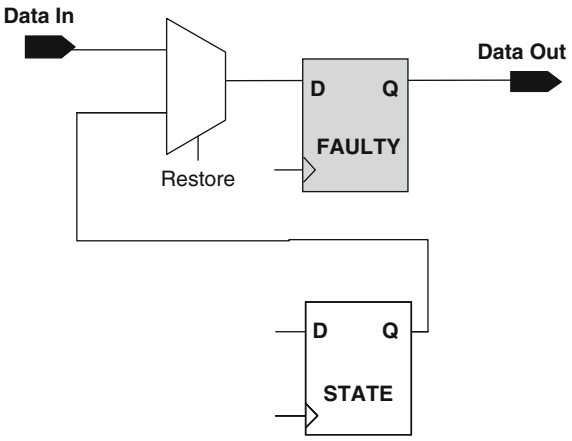


Fig. 6.6 Possible instrument to support injection state restoration in one clock cycle

The obtained benefit will depend on the time required to perform the restoration and, therefore, on the technique used to implement this optimization. Figure 6.6 presents a possible scheme to replace every original flip-flop in order to support the state restoration in just one clock cycle. It includes an additional flip-flop that contains the injection state to be restored in the circuit when the fault is going to be emulated.

Let us suppose that the state restoration requires only one clock cycle, the workload consists of C clock cycles, and the circuit under test contains F sensitive memory elements. Considering that all the memory elements have the same probability to be affected by a fault in any workload cycle, the number of possible single faults is $F{\cdot}C$. In the worst case, with no optimizations, C clock cycles are necessary to emulate each fault. Taking into account all the possible single faults, the total time spent during the fault injection campaign in emulating the circuit without faults is $(1 + 2 + 3 + \cdots + C - 1)$ clock cycles. When the state restoration is performed in just one cycle, this optimization avoids the emulation of C_S clock cycles, where

$$C_S = F\frac{C(C-1)}{2}$$

For example, for a circuit with $F = 10^3$, a workload with $C = 10^5$ clock cycles, and $C_S \sim 0.5 \times 10^{13}$ clock cycles, the total saved time at 100 MHz would be 14 h.

6.4.4 Early Fault Classification

Emulation of a fault finishes when the fault is classified or when the end of the workload is reached. A typical fault classification consists in considering three

categories: failure, latent, and silent faults. In order to detect a failure, the faulty circuit outputs must be compared with the golden behavior. To distinguish between silent and latent faults, internal resources must be observed. Reducing the time to classify faults introduces an important optimization in the evaluation process speed. In general, fault injection techniques stop the fault evaluation as soon as a failure is detected, since outputs observation and comparison with expected values are usually straightforward. Due to the limited observability of the internal resources, in most of the hardware fault injection techniques, classifying a fault as either silent or latent is not feasible or, otherwise the classification is performed at the end of the workload execution, which is very time-consuming. However, silent faults can be detected as soon as the fault effect disappears if the internal elements are observed continuously.

Speeding up silent fault classification requires access to every memory element within the circuit under test in a fast and continuous way, comparing their content with the golden circuit state. Additional hardware is used to store the golden state and to perform the comparison. This extra hardware is shown in Fig. 6.7 and consists of two flip-flops to run the golden and the faulty execution at every workload instant. This optimization is possible in an Autonomous Emulation system with a low cost, since the complete system is implemented in the same hardware device.

Early silent fault classification enhances the fault injection process speed, especially in circuits with fault-tolerant structures that correct or mask faults, where the percentage of silent faults is high. Therefore, applying both optimizations, state restoration (described in the previous section) and early silent fault classification, the time spent in emulating one fault can be drastically reduced (Fig. 6.8).

Putting it all together, [37] describes an instrument to replace every original flip-flop that supports Autonomous Emulation, state restoration, and early silent fault classification. Such instrument is shown in Fig. 6.9. In this case, combinational logic is shared, avoiding the duplication of the complete circuit. Then, the faulty and golden emulation are executed alternately. This implementation for an Autonomous Emulation system is named *Time-Multiplexed* technique.

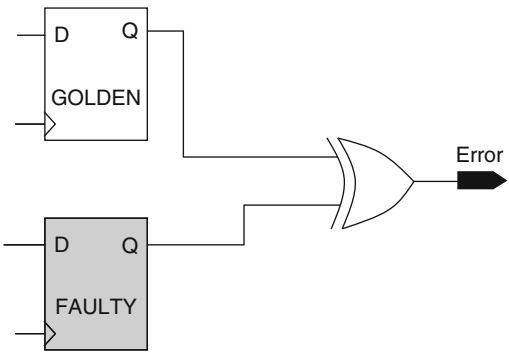


Fig. 6.7 Additional hardware required to implement early fault classification

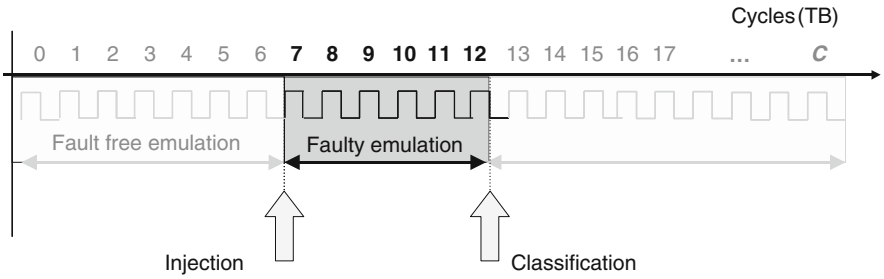


Fig. 6.8 Optimized fault emulation

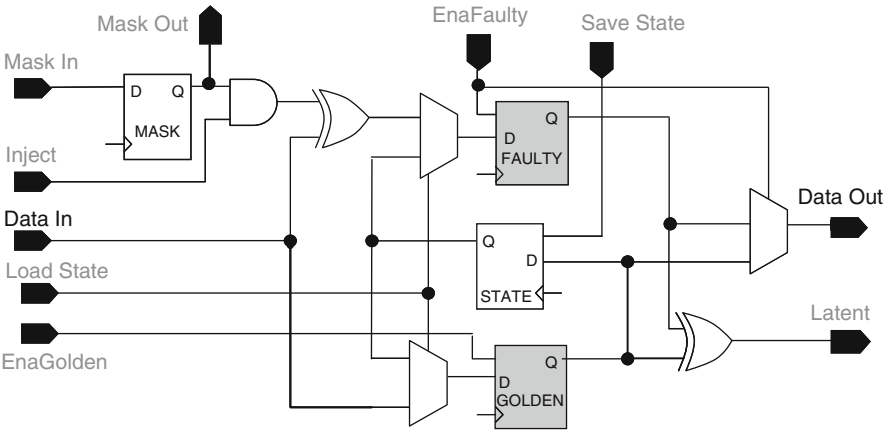


Fig. 6.9 Hardware logic to support all optimizations presented in [37]

For failure and silent faults, the time elapsed between fault injection and fault classification is usually a few clock cycles in circuits with fault-tolerant mechanisms. Only faults with long latencies require the execution of most of the workload, in case of latent faults until the end of the workload. Therefore, if fault latencies and the number of latent faults are small, which is the usual case in a well-defined experiment, the reduction in execution time is proportional to the workload length.

Experimental results reported in [37] show that using the described optimizations, the fault-tolerance evaluation process can achieve fault injection rates in the order of one million faults per second.

6.4.5 Embedded Memories

Embedded memories are common components in modern digital circuits. SRAM memories are sensitive to SEU in the same manner as flip-flops, and then, fault

injection must be performed not only to evaluate flip-flops but also to evaluate effects in the circuit behavior when an SEU affects a memory cell.

Fault injection in embedded memories using emulation-based techniques is a complex task due to the limited observability of the memory cells, since only a memory word can be accessed in a clock cycle. For example, in order to know if there is an error in a memory, the complete memory content should be read, which is very time-consuming.

Few works have been published about emulation-based fault injection in circuits with embedded memories [28, 40, 44, 45]. Civera et al. [44] proposed the fault injection in memory controlling the control and data memory buses to insert a fault in a given memory bit, but it does not propose a solution to analyze faults inside the memory. Lima et al. [40] presents a memory model that consists of a dual-port memory; one port is used to perform the golden emulation, while the other port is used to inject faults. However, the result analysis is very time-consuming since it consists in reading every memory word, comparing obtained data to expected ones.

Portela-García et al. [28] describes an Autonomous Emulation system with optimizations in a circuit with embedded memories. Autonomous Emulation speeds up the injection process by minimizing PC-FPGA communication requirements and including optimizations previously described (state restoration and early silent fault classification). In order to apply the Autonomous Emulation concept in complex circuits, solving the fault injection in embedded memories in a fast and cost-effective way is mandatory.

Autonomous Emulation solution is based in an instrumentation mechanism, so a memory instrument is necessary. It is assumed that embedded memories are synchronous, i.e., they can be prototyped in current FPGAs using the available block RAMs components, and they do not contain useful information before starting the execution. The objective is to achieve a memory model that supports state restoration and early silent fault classification [45].

The proposed solution is based on controlling and observing memory access buses (address, data, and control signals). On the one hand, fault classification requires distinguishing between silent and latent faults. For this purpose, input data bus and control access signals (like write enable, output enable, etc.) in golden and faulty execution are compared (Fig. 6.10). The emulation controller detects the insertion of faulty data in the memory and checks if fault effect is cancelled by writing the correct data during workload execution. As soon as the fault disappears, it is classified as silent.

On the other hand, fault injection to emulate an SEU in a memory cell is performed only in read data, since faults in other memory words do not affect the circuit behavior and they would be classified as latent faults. Therefore, the number of faults to evaluate is reduced significantly.

A possible implementation of the faulty memory in this model consists in storing just the faulty memory words. In order to access Faulty Memory in a fast way, spending just one clock cycle, it is implemented using a Content Addressable Memory (CAM) [45]. Therefore, the faulty memory contains the addresses that store a fault and the faulty data itself. If a given address is stored in the faulty

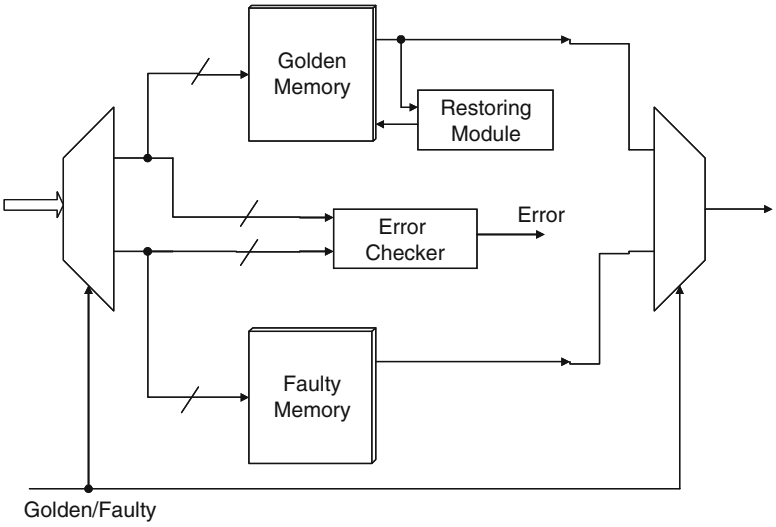


Fig. 6.10 Memory instrument to support Autonomous Emulation in complex circuits with embedded memories

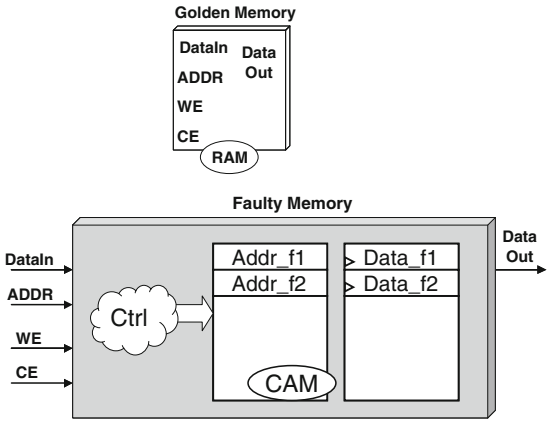


Fig. 6.11 ECAM model is a possible implementation for the memory instrument compliant to Autonomous Emulation

memory, the corresponding memory word contains a fault. Otherwise, the data are stored only in golden memory. This implementation is named Error Content Addressable Memory (ECAM), see Fig. 6.11. ECAM implementation is very suitable to perform the required Autonomous Emulation tasks, such as state restoration or silent fault classification (if faulty memory is empty).

The size of an ECAM implementation fixes the maximum number of errors that can be considered. In practice, the probability that N faults in memory are cancelled (writing a correct value) is very low for just a small value of N. Therefore,

this solution implies less area overhead than other possible implementations (like memory duplication).

Portela-García et al. [28] and Valderas et al. [46] present experimental results on a LEON2 microprocessor. The experiments consist in injecting millions of faults in flip-flops and memories by means of an Autonomous Emulation system.

6.5 Conclusions

Hardware fault injection plays a key role in the design and validation of robust circuits. As hardware reliability is becoming an increasing concern in many application areas, there is a need for new approaches and solutions that can deal with more complex circuits and reproduce fault effects accurately and efficiently.

Hardware fault injection methods have significantly evolved in the last years. Among the physical fault injection methods, accelerated radiation tests are the most used, but laser fault injection has gone through substantial developments. On the contrary, FPGAs can support very efficient logical fault injection methods, such as Autonomous Emulation. These methods can provide unprecedented levels of performance and fault injection capabilities, and represent suitable fault injection mechanisms to complement physical methods.

References

1. R. C. Baumann, "Radiation-Induced Soft Errors in Advanced Semiconductor Technologies", IEEE Transactions on Device and Materials Reliability, Vol. 5, No. 3, pp. 305–316, September 2005.
2. J. Karlsson, P. Liden, P. Dalgren, R. Johansson, U. Gunnelfo, "Using Heavy-Ion Radiation to Validate Fault Handling Mechanisms", IEEE Micro, pp. 8–23, February 1994.
3. S. Duzellier, G. Berger, "Test Facilities for SEE and Dose Testing", Radiation Effects on Embedded Systems. Springer 2007. The Netherlands. pp. 201–232.
4. R. Ecoffet, "In-Flight Anomalies on Electronic Devices", Radiation Effects on Embedded Systems. Springer 2007. The Netherlands. pp. 31–68.
5. IEEE Standard for Environmental Specifications for Spaceborne Computer Modules, March 1997.
6. JEDEC Standard JESD89A, "Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices", October 2006.
7. JEDEC Standard JESD57, "Test Procedures for the Measurement of Single-Event Effects in Semiconductor Devices from Heavy Ion Irradiation", December 1996.
8. S. Buchner, P. Marshall, S. Kniffin, K. LaBel, "Proton Test Guideline Development – Lessons Learned", NASA/Goddard Space Flight Center, NEPP, August 2002.
9. European Space Agency, "Single Event Effects Test Method and Guidelines", October 1995.
10. R. Velazco, S. Rezgui, R. Ecoffet, "Predicting Error Rate for Microprocessor-Based Digital Architectures Through C.E.U. (Code Emulating Upsets) Injection", IEEE Transactions on Nuclear Science, Vol. 47, No. 6, pp. 2405–2411, December 2000.
11. R. Velazco, B. Martinet, G. Auvert, "Laser Injection of Spot Effects on Integrated Circuits", 1st Asian Test Symposium, pp. 158–163, November 1992.

12. P. Fouillat, V. Pouget, D. Lewis, S. Buchner, D. McMorrow, “Investigation of Single-Event Transients in Fast Integrated Circuits with a Pulsed Laser”, International Journal of High Speed Electronics and Systems, Vol. 14, No. 2, pp. 327–339, 2004.
13. F. Miller, N. Buard, T. Carrière, R. Dufayel, R. Gaillard, P. Poirot, J. M. Palau, B. Sagnes, P. Fouillat, “Effects of Beam Spot Size on the Correlation Between Laser and Heavy Ion SEU Testing”, IEEE Transactions on Nuclear Science, Vol. 15, No. 6, pp. 3708–3715, December 2004.
14. D. Powell, J. Arlat, Y. Crouzet, “Estimators for Fault Tolerance Coverage Evaluation”, IEEE Transactions on Computers, Vol. 44, No. 2, pp. 261–274, February 1995.
15. J. Arlat, A. Costes, Y. Crouzet, J. C. Laprie, D. Powell, “Fault Injection and Dependability Evaluation of Fault-Tolerant Systems”, IEEE Transactions on Computers, Vol. 42, No. 8, pp. 913–923, August 1993.
16. H. Maderia et al. “RIFLE: a general purpose pin-level fault injector”, Proceedings of the First European Dependable Computing Conference, Berlin, Germany, October 1994, pp. 199–216.
17. P. Folkesson, S. Svensson, J. Karlsson, “A comparison of simulation based and scan chain implemented fault injection (SCIFI)”, Proceedings of FTCS-28, IEEE Computer Society Press, Munich, June 1998, pp. 284–293.
18. O. Gunnetlo, J. Karlsson, J. Tonn, “Evaluation of error detection schemes using fault injection by heavy-ion radiation”, Proceedings of the 19th Ann. Int’l Symp. Fault-Tolerant Computing, IEEE CS Press, Los Alamitos, CA, 1989, pp. 340–347.
19. J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J. C. Fabre, J. C. Laprie, E. Martins, D. Powell, “Fault Injection for Dependability Validation: A Methodology and some Applications”, IEEE Transactions on Software Engineering, Vol. 16. No. 2, pp. 166–182, February 1990.
20. M. C. Hsueh, T. K. Tsai, R. K. Iyer, “Fault Injection Techniques and Tools”, IEEE Computer, Vol. 30, No. 4, pp. 75–82, April 1997.
21. G. Kanawati, N. A. Kanawati, J. A. Abraham, “FERRARI: A Flexible Software-Based Fault and Error Injection System”, IEEE Transactions on Computers, Vol. 44, No. 2, pp. 248–260, February 1995.
22. J. Carreira, H. Madeira, J. G. Silva, “Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers”, IEEE Transactions on Software Engineering, Vol. 24, No. 2, pp. 125–136, February 1998.
23. T. Jarboui, J. Arlat, Y. Crouzet, K. Kanoun, T. Marteau, “Analysis of the effects of real and injected software faults: Linux as a case study”, IEEE Proceedings of 2002 Pacific Rim international Symposium on Dependable Computing (PRDC’02), 2002.
24. M. Rebaudengo, M. Sonza Reorda, “Evaluating the Fault Tolerance Capabilities of Embedded Systems via BDM”, 17th IEEE VLSI Test Symposium, pp. 452–457, Dana Point, USA, April, 1999.
25. IEEE-ISTO 5001-2003, “The Nexus Forum™ standard for a global embedded processor debug interface”, version 2.0, 2003.
26. A. V. Fidalgo, G. R. Alves, J. M. Ferreira, “Real Time Fault Injection Using Enhanced OCD – A Performance Analysis”, 21st IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT), 2006.
27. J. Peng, J. Ma, B. Hong, C. Yuan, “Validation of Fault Tolerance Mechanisms of an Onboard System”, 1st International Symposium on Systems and Control in Aerospace and Astronautics (ISSCAA), pp. 1230–1234, January 2006.
28. M. Portela-García, M. García-Valderas, C. López-Ongil, L. Entrena, “An Efficient Solution to Evaluate SEU Sensitivity in Digital Circuits with Embedded RAMs”, XXI Conference on Design of Circuits and Integrated Systems (DCIS’06), November 2006.
29. P. Kenterlis, N. Kranitis, A. Paschalis, D. Gizopoulus, M. Psarakis, “A Low-Cost SEU Fault Emulation Platform for SRAM-Based FPGAs”, 12th IEEE International On-Line Testing Symposium, pp. 235–241, July 2006.
30. M. Alderighi, F. Casini, S. D’Angelo, M. Mancini, A. Marmo, S. Pastore, G. R. Sechi, “A Tool for Injecting SEU-like Faults into the Configuration Control Mechanism of Xilinx

Virtex FPGAs", 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2003.
31. M. Alderighi , F. Casini, S. D'Angelo, M. Mancini, S. Pastore, G. R. Sechi, R. Weigand, "Evaluation of Single Event Upset Mitigation Schemes for SRAM Based FPGAs Using the FLIPPER Fault Injection Platform", 22nd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 105–113, 2007.
32. K. T. Cheng, S. Y. Huang, W. J. Dai, "Fault emulation: a new approach to fault grading", Proceedings of the International Conference on Computer-Aided Design, pp. 681–686, 1995.
33. L. Antoni, R. Leveugle, B. Feher, "Using Run-Time Reconfiguration for Fault Injection in HW Prototypes", IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 245–253, 2002.
34. M. Aguirre, J. N. Tombs, F. Muñoz, V. Baena, A. Torralba, A. Fernandez-Leon, F. Tortosa, D. Gonzalez-Gutierrez, "An FPGA based hardware emulator for the insertion and analysis of Single Event Upsets in VLSI Designs", Radiation Effects on Components and Systems Workshop, September 2004.
35. J. H. Hong, S. A. Hwang, C. W. Wu, "An FPGA-Based Hardware Emulator for Fast Fault Emulation", MidWest Symposium on Circuits and Systems, 1996.
36. P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, "Exploiting Circuit Emulation for Fast Hardness Evaluation", IEEE Transactions on Nuclear Science, Vol. 48, No. 6, 2001.
37. C. López-Ongil, M. García-Valderas, M. Portela-García, L. Entrena, "Autonomous Fault Emulation: A New FPGA-based Acceleration System for Hardness Evaluation", IEEE Transactions on Nuclear Science, Vol. 54, Issue 1, Part 2, pp. 252–261, February 2007.
38. M. Violante, "Accurate Single-Event-Transient Analysis via Zero-Delay Logic Simulation", IEEE Transactions on Nuclear Science, Vol. 50, No. 6, December 2003.
39. M. García Valderas, R. Fernández Cardenal, C. López Ongil, M. Portela García, L. Entrena. "SET emulation under a quantized delay model", Proceedings of the 22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFTS), pp. 68–78, September 2007.
40. F. Lima, S. Rezgui, L. Carro, R. Velazco, R. Reis, "On the use of VHDL simulation and emulation to derive error rates", Proceedings of 6th Conference on Radiation and Its Effects on Components and Systems (RADECS'01), Grenoble, September 2001.
41. F. Faure, P. Peronnard, R. Velazco, R. Ecoffet, "THESIC+, a flexible system for SEE testing", Proceedings of RADECS Workshop, [September 19–20, 2002, Padova], pp. 231–234.
42. D. Lewis, V. Pouget, F. Beaudoin, P. Perdu, H. Lapuyade, P. Fouillat, A. Touboul, "Backside Laser Testing of ICs for SET Sensitivity Evaluation", IEEE Transactions on Nuclear Science, Vol. 48, Issue 6, Part 1, pp. 2193–2201, December 2001.
43. F. Faure, R. Velazco, P. Peronnard, "Single-Event-Upset-Like Fault Injection: A Comprehensive Framework", IEEE Transactions on Nuclear Science, Vol. 52, Issue 6, Part 1, pp. 2205–2209, December 2005.
44. P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, "FPGA-Based Fault Injection for Microprocessor Systems", IEEE Asian Test Symposium, pp. 304–309, 2001.
45. M. Nicolaidis, "Emulation/Simulation d'un circuit logique", French patent, filed February 25 2005, issued October 12 2007.
46. M. G. Valderas, P. Peronnard, C. Lopez-Ongil, R. Ecoffet, F. Bezerra, R. Velazco, "Two Complementary Approaches for Studying the Effects of SEUs on Digital Processors", IEEE Transactions on Nuclear Science, Vol. 54, Issue 4, Part 2, pp. 924–928, August 2007.

Chapter 7
Integrated Circuit Qualification for Space and Ground-Level Applications: Accelerated Tests and Error-Rate Predictions

Raoul Velazco, Gilles Foucard, and Paul Peronnard

Integrated circuits (analog, digital or mixed) sensitivity evaluation to Single Event Effects (SEE) requires specific methodologies and dedicated tools. Indeed, such evaluation is based on data gathered from on-line tests performed in a suitable facility (cyclotron, linear accelerator, laser , etc.). The target circuit is exposed to particles fluxes having features (energy and range in Silicon) somewhat representative of the ones the circuit will encounter in its final environment. This chapter will describe and illustrate with experimental results, the methodologies and the hardware and software developments required for the evaluation of the sensitivity to SEE of integrated circuits. Those techniques will be applied to a SRAM-based FPGA and to a complex processor. In case of sequential circuits such as processors, the sensitivity to SEE will strongly depend on the executed program. Hardware/software fault-injection experiments, performed either on the circuit or on an available model, are proved as being complementary of radiation ground testing. Indeed, data issued from fault injection combined with data issued from radiation ground testing allow in accurately predicting the error rate of any program.

7.1 Introduction

Constant progress of manufacturing technologies and processes, results in an increase of integrated circuits sensitivity to the effects of natural radiation. This problem, which in the past years was a concern only for space applications, must nowadays be considered for any application requiring reliability or dependability, devoted to operate in earth's atmosphere, even at ground level. Indeed, an energetic particle (heavy ion, neutron, proton, etc.) hitting a sensitive area of a circuit may

R. Velazco (✉), G. Foucard, and P. Peronnard
TIMA Labs, 46 Av. Félix Viallet, 38031 Grenoble, France
e-mail: raoul.velazco@imag.fr; gilles.foucard@imag.fr; paul.peronnard@imag.fr

M. Nicolaidis (ed.), *Soft Errors in Modern Electronic Systems*,
Frontiers in Electronic Testing 41, DOI 10.1007/978-1-4419-6993-4_7,
© Springer Science+Business Media, LLC 2011

induce failures with potentially serious consequences. Among radiation's effects on integrated circuits, the so-called SEU (Single Event Upset) phenomenon is considered critical as its consequence is a memory cell content modification randomly in time and location [1, 2].

At the system level, an SEU, also called bit-flip, upset or soft error, may propagate with different kinds of consequences depending on the final application criticality. This makes the system error-rate evaluation a mandatory task. To achieve such an evaluation, usually adopted strategies are based on radiation ground testing results, also called *accelerated* tests. Those experiments are performed in facilities such as cyclotrons or linear accelerators and consist in exposing target circuits to suitable particle beams, while they are performing an activity considered as representative of the one they will carry out in the final application.

As present applications require more and more data processing power, this entails the use of complex advanced integrated circuits, such as processors and reconfigurable circuits. The complexity of these devices makes the evaluation of sensitivity to soft errors difficult to achieve due to the constant progress of manufacturing technologies. Indeed, hardware and software requirements needed to set up radiation ground testing experiments and high costs related to the use of particle accelerator facilities to simulate radiation environment make this kind of testing reserved to few specialized research and development teams. Moreover, any change in the application software would imply extra radiation ground testing campaigns, thus increasing the development costs.

In this chapter, ionizing radiation effects on integrated circuits are briefly reviewed. Test methods and tools required to perform experiments, using suitable radiation facilities in which the target circuit is exposed to representative radiation beams, are described. The test methodology must be adapted to the target circuit nature. Indeed, advanced complex digital circuits such as processors cannot be tested using the same methods than the ones required for Application Specific Integrated Circuits (ASICs), memories (SRAMs, DRAMs, MRAMs, etc.), or reconfigurable circuits such as Field Programmable Gate Arrays (FPGAs). Case studies will illustrate the different methods applied to representative circuits.

Most of digital system architectures include processors (or processors implemented by means of FPGAs). For these architectures, a methodology will be described that allows to predict SEU error rates by combining results issued from radiation ground testing with those obtained from hardware/software fault injection experiments performed off-beam. The goal of this strategy is to obtain accurate results about different applications' error rates without using particle accelerator facilities, thus significantly reducing the cost of such tests. This will be illustrated for a complex processor, the Power PC 7448, executing a program issued from a real space application.[1] Results are confronted to the ones obtained from exposure to radiation of the same architecture and configuration.

[1] An AOCS (Attitude and Orbit Control Subsystem) module provided by the French Space Agency CNES (Centre National d'Etudes Spatiales).

7.2 Single Event Effects Due to Radiation and Their Consequences on Integrated Circuits

Energetic particles hitting Silicon lose energy along their tracks. This energy is absorbed by the matter creating electron-hole pairs. Their charge may be collected by diffusion and drift mechanisms resulting in a spurious current pulse at the impacted node. This current pulse has a duration and amplitude which depend on both the incident particle features (energy and mass) and the circuit's technology. The phenomena caused by this current pulse are gathered under the acronym SEE. SEEs are generally classified according to their consequences. Destructive SEEs include Single Event Latchup (SEL), Single Event Gate Rupture (SEGR), and Single Event Burnouts (SEB). Among non-destructive SEEs can be mentioned Single Event Upset (SEU), Multiple Bit Upset (MBU), Multiple Cell Upset (MCU), and Single Event Transient (SET). In Chapter 2 are described in detail all these events. References 1 and 2 address the consequences of these events.

A SET is a spurious current pulse propagating into combinational nodes and may become an SEU if, when captured, it alters the content of a memory cell (flip-flop, latch, etc.).

SEUs occur when the current pulse resulting from ionization changes one of the considered circuit's memory cell content. Its consequences depend on both the corrupted information nature and the occurrence instant. For complex devices such as processors, FPGAs, etc., the SEU occurrence may lead to a critical misbehavior. Such phenomenon called SEFI (Soft Error Functional Interrupt) can only be recovered by a soft or hard reset.

With the transistor's dimensions shrinking, the charge collected after a particle strike may be distributed to several adjacent nodes, leading to MBU if perturbed bits are in the same memory word or to MCU in other cases.

A SEL occurs when the collected charges provoke a short circuit between ground and power by triggering the parasitic thyristor present in all CMOS circuits. Single Event Latch-ups can be detected by measuring on-line the circuit (or system) power consumption, and shutting down the power supply if the consumption is higher than a predetermined limit. At the opposite, consequences of SEUs might be difficult to detect or predict, as they depend on the future use of the modified information after a SEU occurrence.

This rough analysis of the consequences of spurious modifications of complex circuits' memory cell contents aims at showing that the SEU phenomenon consequences are potentially critical for any application devoted to operate in space, or even in the earth's atmosphere, and must thus be considered with care while selecting parts and developing the system architecture.

SEEs can be mitigated by several techniques performed at different levels [3]:

- Technology process: SOI (Silicon On Insulator) eliminates SEL and reduces SEU rate.
- Design process: P+ guard rings for SEL. Replacement of memory cells by hardened ones deals with SEU, thus providing time or space redundancy.

- Software level: Error correcting and/or detecting codes.
- System level: HW/SW redundancy.

To ensure a system meets its dependability requirements, various types of experiments are mandatory. These experiments, when related with SEE must be performed in particle accelerator facilities (cyclotron, linear accelerators, etc.). Methodologies and tools needed to achieve such experiments are described in the following chapters.

7.3 Accelerated Tests: Methodologies and Related Results

SEE radiation ground testing consists in exposing the Device Under Test (DUT) to energetic particles (neutrons, heavy ions, protons, etc.), while it performs an activity which strongly depends on the experimenter expected results. Thus, such tests are performed on-line, i.e., the DUT is powered and active in its nominal operating conditions. The main goal of the accelerated test is the evaluation of the device sensitivity to radiation in terms of the particles average number required to provoke an event. This measure is called cross-section and is obtained from a static test approach.

At a system level, the DUT individual sensitivity contribution requires performing the so called dynamic testing, i.e., the target device which is exposed to the beam will be implemented in its final hardware/software environment. Only SEU faults will be taken into account in the following.

7.3.1 Notion of Cross-Section

SEEs' occur when a particle hitting a circuit meets the following conditions [4]:

- It has the sufficient energy and it will deposit the minimum charge required to generate the considered event.
- The impact takes place within a sensitive area (drain of an off-transistor).

Radiation test facilities such as cyclotron, linear accelerator, etc., provide various beams (neutrons heavy ions, protons, etc.), allowing the analysis of the SEE impact on micro-electronic devices. However, these particles are issued as a broad beam, thus they do not allow studying precisely the impact of a single particle directed on a selected node of the circuit. Moreover, the DUT has many sensitive volumes, and so it is not possible to know which particle beam will interact with a given device-sensitive volume. As a consequence, radiation ground testing data have a statistical nature, and are function of the beam features (flux and particle energy), and the device features: Sensitive volume (SV) size, number of SV per surface unit. The cross-section concept allows removing geometrical parameters and provides a particle/device dynamic interaction measure.

Let us imagine an interaction between an incoming energetic particle and a device with a sensitive volume array. Moreover, let us assume that the particle strikes the device with a normal incidence and has enough energy to depose a charge that is able to trigger a single event if it passes through the sensitive volume. The energy transferred to the silicon is called LET[2] (Linear Energy Transfer). The end-result of this experiment is the single event rate by unit of time (R_b). This virtual set-up is given in Fig. 7.1.

We can forecast that we will be able to observe a single event each time an impinging particle enters the top face of a sensitive volume (noted σ). If the device surface seen by the beam is S, and N_b is the number of sensitive volumes in S, the total sensitive surface of S is $\sigma \times N_b$ and we have

$$\frac{\sigma \times N_b}{S} = \frac{R_b}{R_a},$$

R_a is the incoming particle number crossing the surface S by unit of time, so

$$\sigma = \frac{R_b}{R_a/S \times N_b} = \frac{R_b}{\phi a \times N_b},$$

where $\Phi a = R_a/S$ is the beam flux (number of particles per unit of surface and time).

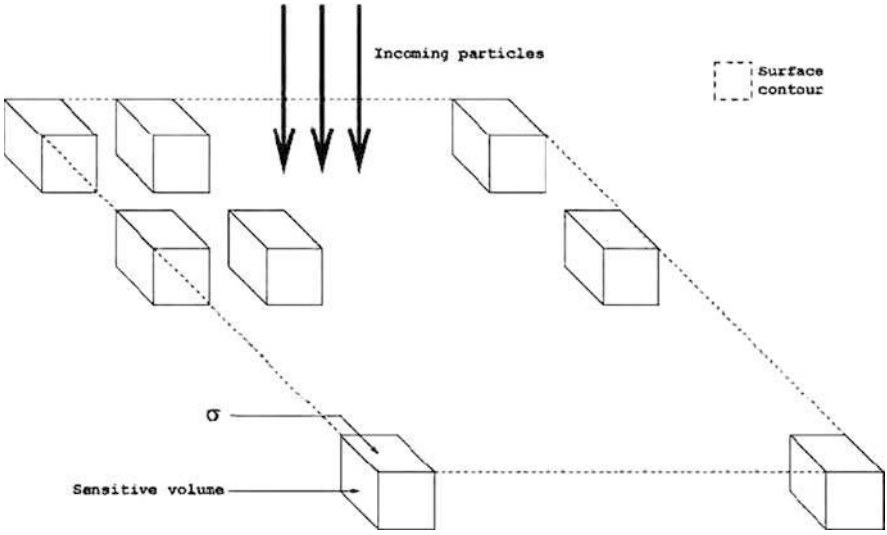


Fig. 7.1 Virtual experiment set up

[2]The LET (Linear Energy Transfer) is the ionization measurement caused as the incoming particle looses energy along its path. For instance, in Silicon, a particle having a LET of 97 MeV cm^2/mg is depositing about 1 pC/μm.

If we introduce the experiment duration T,

$$\sigma = \frac{R_b \times T}{\phi a \times N_b \times T}$$
$$= \frac{\text{Number of single event recorded during the experiment}}{\text{Fluence of the experiment} \times \text{number of sensitive volumes exposed}}.$$

σ is called the interaction cross-section and is a direct measure of the sensitive volume top surface. Its unit is cm^2 or it is expressed in barn (1 barn $= 10^{-24}$ cm^2). Often the number of sensitive volumes is not known, and σ is given as an interaction cross-section per device (or per bit).

$$\sigma_{dev} = \frac{\text{Number of single event recorded during the experiment}}{\text{Fluence of the experiment}}.$$

$$\sigma_{bit} = \frac{\text{Number of single event recorder during the experiment}}{\text{Fluence of the experiment} \times \text{number of bits}}.$$

As a consequence, the SEU radiation ground testing end-product will be a plot of the interaction cross-section vs. particles' LETs. In a typical cross-section curve, such as the one given in Fig. 7.2, two important magnitudes must be noticed: the LET threshold,

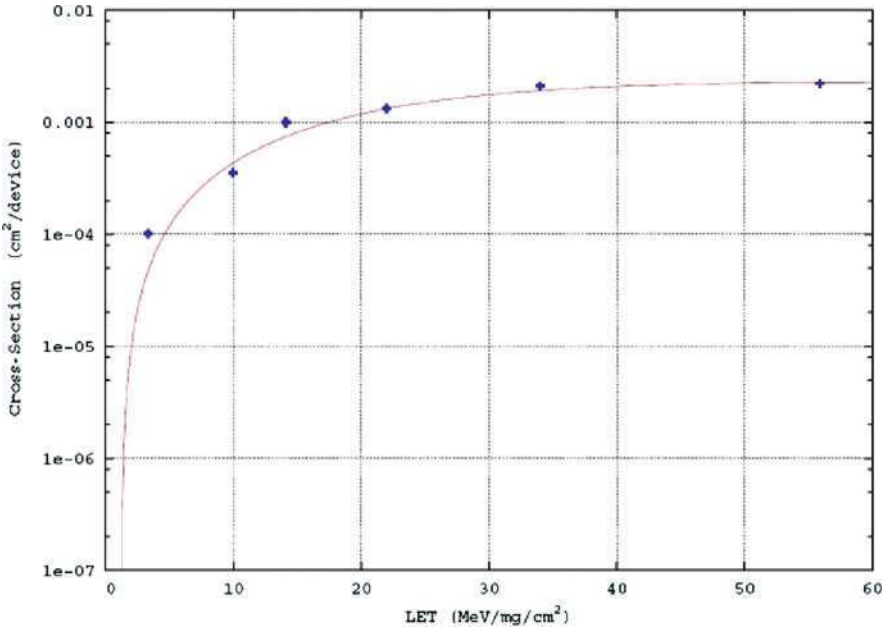


Fig. 7.2 A typical cross-section curve

Table 7.1 Environments to be assessed vs. LET threshold

Device sensitivity threshold	Environment to be assessed
LETth $<$ 10 MeV cm^2/mg	Cosmic rays, trapped protons, solar flares
LETth $=$ 10–100 MeV cm^2/mg	Cosmic rays
LETth $>$ 100 MeV cm^2/mg	No analysis required

which is the first LET provoking an event, and the saturation cross-section, which corresponds to the whole sensitive area.

From the LET threshold and the final environment nature, a decision can be made regarding the assessment needed to use the tested circuit in a reliable way. Table 7.1 summarizes LET threshold typical values and their corresponding assessment. The LET threshold obtained measures allow in determining a first diagnosis about the DUT that suitably fits the final radiation environment with the requirements.

To derive the SEU cross-section, the circuit must be active during its exposure to the selected particle beam. One important methodological aspect is the choice of the activity the DUT will achieve during the radiation experiment. This strongly depends on the circuit type, being quite straightforward for circuits having a single function, such as memories, and quite difficult for circuits, such as processors, capable of executing a set of instructions or commands. The next section will briefly expose some trends about main strategies usually adopted for representative circuits.

7.3.2 Static and Dynamic SEU Test Strategies

For chips such as memories (SRAMs, DRAMs, etc.), the sensitivity estimation to SEUs can easily be done by loading them with a predetermined pattern, whose corruption by SEUs is identified by periodically reading out the target memory area and comparing it with expected values.

Such a strategy is generally called *static test*, because all accessible targets are exposed at the same time to SEUs, a situation which is far from the real circuit activity. When used during radiation ground testing, a static test strategy will provide worst-case memory sensitivity estimation to SEUs. Indeed, when included into an equipment, the memory will be used to store information. Some of them (a program code or a data area for instance) stay unmodified during the whole system operation, but some other zones are used more dynamically. A way to determine a sensitivity closer to one of the final equipment consists in evaluating the average number of used bits and estimating the sensitivity as the product of this number by the per-bit sensitivity (derived from the memory cross-section issued from radiation testing divided by the number of bits).

When the DUT is a processor, attempts of adopting the static strategy were made by considering the target memory area accessible by the instruction set. This area includes all DUT registers and internal memory. A program implementing a static test strategy can be easily written using mainly LOAD and STORE instructions targeting all those processor's memory elements accessible to the instruction set

(registers, internal SRAM, etc.). Running this program when the processor is exposed to radiation will provide a worst-case estimation of the cross-section, which does not correspond to the real DUT sensitivity when running a real application. Indeed, registers and various memory zones are used in a dynamic and uncontrollable manner when running a given program: data stored in a register, for instance, will be sensitive to radiation only during the period the data will have a meaning for the program.

Extrapolating memory testing strategy to processors was shown to significantly overestimate the error rate. Indeed, in [5, 6], authors performed ground testing with both a static test strategy (in which the program executed by the processor under test is restricted to loading and observing the register set and internal memory) and a set of programs intended to be somewhat "representative." Application's results of such an approach to different processors, including 16-bit processors (the Motorola 68000 and 68882) and a reduced instruction set processor with parallel capabilities (the Inmos Transputer T400), showed that measured error rates for those programs and processors were between one or two orders of magnitude less than those derived from ground tests performed with static test strategy.

Those results are not surprising. Indeed, to induce a wrong result at a program output, one or more upsets must occur during the target sensitive period. Moreover, many registers and memory elements are not used by the program; upsets affecting those latches will have no effect on the program outputs, and consequently will not contribute to the final error rate. Unfortunately, there is little published work comparing error rates derived from ground testing using benchmark programs to error rates obtained either during ground testing or observed in flight.

Ideally, the final application program should be used during radiation experiments to get realistic cross-section measurements, but often it is not available at the time of the radiation test campaign. An interesting approach to deal with this difficulty was proposed in [7]. In this study, the authors stress the importance of evaluating the application's cross-section and calculate the *duty factors* for various processor elements used by the software. The time between loading a given register or memory location and writing it to memory is the period during which the register is sensitive to upsets. When this period is expressed as a percentage of the total execution time, it is called the *duty factor* of that particular register or memory element.

In this way, calculating a program error rate will provide good estimations requiring limited ground test experiments in order to calculate the DUT cross-section. Duty cycles calculated from an analysis performed on assembly language source code can be used to follow the future evolution of any application program. Nevertheless, the main difficulty comes from the diversity of processors and instruction sets, making it very difficult in practice to develop a tool automating the duty period calculation. Indeed, since the first presentation of this approach in 1988, no automated tool allowing duty factor derivation for a given application was published in the literature. The development of such a tool is seriously impeded by the combinatorial explosion of execution paths (induced by software construction such as loops and conditional statements) making the analysis intractable.

Moreover, variations in program inputs can result in different instruction paths, different duty factors, and thus different SEU vulnerability. Another limitation comes from the fact that not all SEU targets are accessible through the instruction set. Many temporary registers and flip-flops existing in a processor are not known at the architectural level, and thus their contribution to the global error rate cannot be included in the calculations. Despite this intrinsic limitation, the error rates by calculated taking into account duty periods of SEU accessible targets are generally considered as being quite close to the expected error rates when the circuit operates in the final application.

7.4 Test Facilities: Heavy Ion, Neutron, Proton Accelerators, and Laser

Actual available particle accelerator facilities are not able to reproduce the exact space environment because of its wide particle species distribution, high energies, and multi-directional incidence. Galactic cosmic rays in space have energies distributed with a peak around a few 100 MeV/nucleon, but ions of interest for the SEU issue cover the 0.1–100 MeV/mg/cm^2 LET domain. Only few accelerators are capable of delivering such high energy particles. However, "low" energy beams can reproduce an equivalent charge deposition by means of LET's parameter.

Unlike heavy ions, protons with energies similar to the ones observed in space are available for accelerated ground test experiments.

Next paragraphs briefly present a non-exhaustive list of representative particle accelerators available in Europe for integrated circuits and systems radiation ground testing [8].

7.4.1 Heavy Ions

In Table 7.2 are given the most commonly used heavy ion facilities subdivided in three groups according to their energy ranges.

Due to its high energy beam, the GANIL facility allows "in air" irradiation, but the disadvantage is a long beam setup time. For the other facilities, tests are performed in vacuum chambers.

Table 7.2 Facilities available in Europe for heavy ion ground testing

High energy (100 MeV/amu)	France – GANIL
Medium energy ($\geq$10 MeV/amu)	Belgium – CYCLONE/Finland – JYFL
Low energy ($\leq$10 MeV/amu)	France – IPN/Italy – LNL

7.4.2 Protons

In Table 7.3 is given a list of European proton facilities. As energy losses by protons in air are low, all irradiation are performed in air avoiding vacuum system complexity.

7.4.3 Neutrons

Neutron tests, which mainly concerned the avionics community in the past years, are presently becoming a major concern for any electronic equipment built from advanced sub-micron chips, devoted to operate in the atmosphere. The JEDEC JESD89 [9] specification states that the preferred facility to perform neutron tests is WNL (Los Alamos – USA), because its energy spectrum matches the neutron energy spectrum at sea level. However, the same regulation explains how to obtain data using the quasi-mono energetic neutron beams available in Europe. Two accelerators in Europe are equipped with neutron beam lines: CYCLONE (Belgium) and Svedberg Laboratory (Uppsala, Sweden).

7.4.4 Micro-Beam and Laser

SEE characterization using particle beams (heavy ions, protons, or neutrons) is a global approach. Such test provides a number of events for a particular fluency, without any information about the detected faults location. Testing with laser or micro-beam overcomes this problem. A small spot and precise beam localization allow sensitive device nodes to be pinpointed with submicron accuracy. Similar to that for heavy ion macro-beam tests, experiments must be placed in a vacuum chamber for micro-beam irradiation. Although the laser is not affected by such a problem, it has the disadvantage of having its beam reflected by metallization layers, thus complex circuits must be irradiated from the back side. Another laser limitation is that its energy cannot be correlated with the ions energy. Table 7.4 depicts advantages and drawbacks of heavy ion macro- and micro-beams and laser beams.

Table 7.3 Facilities available in Europe for proton ground testing

CYCLONE – Belgium	Up to 70 MeV
JYFL – Finland	Up to 45 MeV
CPO – France	Up to 200 MeV
IPN – France	Up to 20 MeV
SIRAD – Italy	Up to 28 MeV
PSI/OPTIS – Switzerland	Up to 63 MeV
PSI/PIF – Switzerland	Up to 300 MeV

Table 7.4 Comparison of three different types of beams

	Heavy ion beams	Laser	Heavy ion micro beam
Beam diameter on DUT	A few centimeter	Down to 1 μm	Down to 1 μm
Limitations	Range several tenths of micrometer	Small penetration depth	Range several tenths of micrometer
Localization of sensitive areas	No	Yes	Yes
Study of rare phenomena	No	Yes	Yes
Cross-section determination	Yes	No	Yes

In Europe, only one facility is equipped with a micro-beam: GSI (Darmstadt, Germany). This facility uses its linear accelerator to produce ions from Carbon to Uranium with energies between 1.4 and 11.4 MeV/amu. These kinds of facilities are not suitable to characterize a device (measure of the cross-section sensitivity curve) but they are very useful to help in understanding failure modes.

7.5 Required Test Platforms and Description of a Generic Testbed

7.5.1 Introduction

Evaluating the error rate of equipments designed to operate during radiation exposure is a major concern. A general approach for such estimations is based on both the final environment features and the sensitivity to radiation of parts included in the system. Among those parts, memories and processors present in almost all complex digital systems can be identified as major contributors to the global error rate. Indeed, the large number of memory elements they include makes them potentially sensitive to SEUs, which as stated before is one of the critical effects. As a consequence of a single particle impact, either the data having crucial importance for the application can be corrupted, or the processor program sequencing may be lost, thus provoking unexpected and possible critical system behaviors.

As stated before, estimating the circuit sensitivity to SEE phenomena needs to expose the target circuit to suitable particle fluxes using radiation facilities such as particle accelerators (cyclotrons, linear accelerators, etc.). To provide incident particles' energy precise measurements, those tests are usually performed in vacuum chambers.

The main difficulty entailed by such tests, usually called *accelerated radiation tests*, is the fact that the circuit under test must be active during the experiment duration. This means that the DUT must operate within a typical electronic/digital environment allowing both to make the circuit capable of executing the selected program and detecting the SEEs' effects. In case the DUT is a processor, this includes external memory and all required circuits.

From a hardware point of view, a specific test platform must be developed. In addition to the above-mentioned capabilities, such a platform should provide means of communication for storing the detected error information in an external PC for the purpose of further analysis. A tester based on a motherboard/daughterboard architecture can be designed to reuse some functionalities with different DUTs. The following requirements must be taken into account to design the tester:

- Control the DUT power consumption to detect SELs and switch off the power supply in order to avoid DUT damages.
- Start and stop daughterboard activity; detecting and recovering from SEUs; and provoking processors program sequence-loss (by means of a programmable timer usually called *watchdog*).
- Detect SEUs on-line and recover related information. This last aspect is achieved for those SEUs resulting in an erroneous DUT output, by reading the content of a memory space shared by the motherboard and the DUT board, in which the DUT is supposed to write the program execution results. SEUs resulting in program sequence-loss are detected by means of a timer, which will indicate that the program duration is different than the expected one, and stop the program execution by asserting an asynchronous interrupt signal.

7.5.2 The ASTERICS Test Platform

7.5.2.1 Overview

The ASTERICS (Advanced System for TEst under Radiation of Integrated Circuits and Systems) is a major evolution of the THESIC + test platform presented in [10]. The idea is to implement the digital environment needed by DUTs, by means of an FPGA whose configuration is obtained from compiling this environment description in a hardware description language such as Verilog or VHDL. This way, the user task is limited to wire the relevant DUT signals to the tester ones, so as to adapt the DUT to the test platform. This test platform's architecture was successfully used to qualify many kinds of DUTs such as processors (PowerPC, SPARC, etc.), memories, and mixed analog-digital circuits.

7.5.2.2 Technical Description

In the past, different versions of test systems devoted to radiation ground test were prototyped [11] with the following characteristics:

- The DUT operates in its standard digital environment, i.e., it is interfaced to a suitable architecture. In case of a processor, such architecture involves memories to store test program instructions and data, glue logic, clock, and power supply.
- A daughterboard connected to the testbed embeds the DUT.

- The whole system communicates with the user through a computer by means of a serial or Ethernet link.

As an example, the following describes the ASTERICS architecture, which is based on the same principles, but takes into account the high performance requirements of nowadays ICs. This platform is built around two Xilinx Virtex4 FPGAs. One named Control FPGA and the other named Chipset FPGA. Figure 7.3 depicts this architecture.

- Control FPGA: A Xilinx Virtex4 FX, embedding a PowerPC processor in charge of the communication between the user and the tester via a gigabit Ethernet link. The PowerPC runs at 300 MHz allowing a good data rate transfer.
- Chipset FPGA: A Xilinx Virtex4 FX, optimized for logic resources, embeds the user's design needed by the DUT to operate in its rated condition. In case of a processor, this chipset acts as a memory controller. This FPGA allows operating frequencies about 200 MHz depending on the user's design complexity.

The DUT is connected to the tester via a high-speed connector. Up to 180 I/Os are available with standard selectable by software LVCMOS voltages ranging from 1.2 to 3.3 V. Obviously, different kinds of memories (SRAM and DDR-SDRAM) are available to match different data rate requirements. For latchup sensitive circuits, a current monitoring circuit protects the DUT against destructive latchup phenomena. Figure 7.4 gives a tester overview.

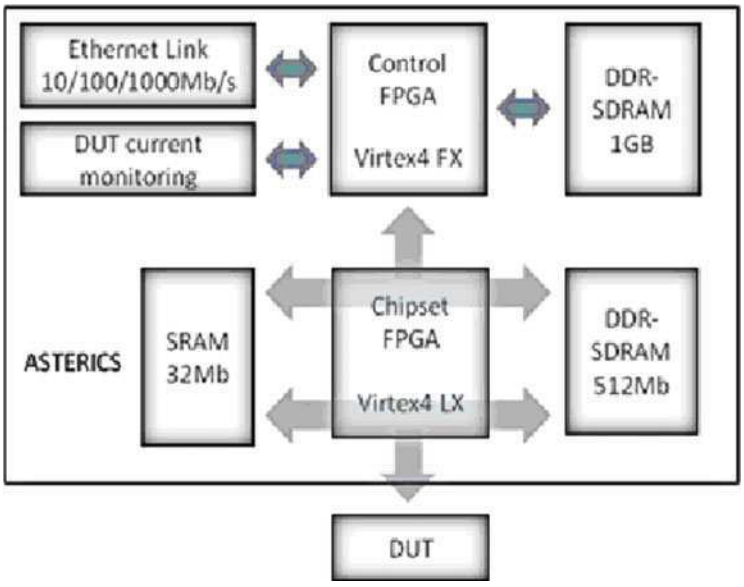


Fig. 7.3 ASTERICS architecture

Fig. 7.4 ASTERICS testbed

To facilitate communication with the tester, an API (Application Programming Interface) is user available. This API embeds a set of functions like memory read/write, setting the DUT current consumption limit, resetting the DUT, etc.

The tester Ethernet link allows users to remotely control experiments anywhere in the world whenever an internet connection is available.

7.6 Radiation Ground Testing: Case Studies

In this section are briefly described the methodologies to perform radiation ground testing on typical advanced digital circuits. SRAMs, FPGAs, and processors are considered to illustrate hardware/software developments required to perform radiation ground testing.

7.6.1 SRAM Memory

Radiation ground testing methods for memories are quite straightforward. Indeed, it consists in loading the memory with a predefined pattern. The component is then left under the beam and its memory content is periodically checked to detect upsets created by impinging particles. Implementation of such a test can be done either by a specific hardware or by adapting existing test platforms like the one described in the previous section.

7.6.2 Processor and Microcontrollers

SEU sensitive area in processors-like devices may include registers, internal SRAM, and data and instruction memory caches. Generally, the used approach is based on the static cross-section evaluation, from which the dynamic cross-section

of any application can be estimated by fault injection experiments, as described in the following section. The methodology to perform SEU radiation ground testing on two generations of an advanced Power PC processor is hereafter described for both heavy ion and neutron tests.

7.6.2.1 Heavy Ions Tests

Heavy Ion Test Conditions

Tested Devices

Heavy ion tests were performed on two Power PC 7448 devices issued from two different diffusion lots.

Considering the DUTs are "flip chip" packaged, i.e., the devices' active zone is facing the solder side, to reach the sensitive areas, the particle beam must pass through the substrate. This obliges to thin the DUT's substrate in order to allow the particle beam deposing enough energy for provoking SEEs in the DUT. Tested devices were thinned down to 70 μm and then tested to guarantee their correct operation.

Test Conditions

In Table 7.5 are summarized the test conditions adopted for heavy ion experiments. The test strategy applied here consists in initializing with predetermined patterns all memory elements accessible through the instruction set. After this step the processor enters in a waiting loop whose duration can be adapted to the tested memory elements sensitivity to guarantee a good statistic of detected events. Generally used patterns are alternating "0" and "1" called Checkerboard (CHB), the inverted Checkerboard (CHBn), the "all at 0" or "all at 1" patterns known, respectively, as All 0 and All 1.

To avoid significant thermal dissipation, clock frequencies were set at 33 MHz. The core frequency was 600 MHz, which is the rated operating frequency.

The used static test program targets the Power PC accessible sensitive regions: The register file (8,928 bits) and the memory data cache L1 (262,144 bits).

Table 7.5 Test conditions for heavy ion experiments

	POWER PC 7448
Number of tested devices	Issued from two different diffusion lots
Core power supply	1 V
I/Os power supply	2.5 V
Temperature	Room temperature
Type of test	Static
Test pattern	CHB, CHBn, All0, All1
Tilt angle	0 and 45°
Operating frequencies	33 MHz SYSCLK, 18× for the multiplier, ~600 MHz for the core

The following registers were tested:

General purpose registers: GPR2 → GPR31	32 × 32 = 1,024 bits
Floating point unit registers: FPR0 → FPR31	32 × 64 = 2,048 bits
Special registers: SPRG0 → SPRG7	8 × 32 = 256 bits
Virtual memory: IBAT0 → IBAT7, DBAT0 → DBAT7	16 × 32 = 512 bits
State registers: SR0 → SR15	16 × 32 = 512 bits
Vector operation registers: VR0 → VR31	32 × 128 = 4,096 bits
14 registers used for fault injection in L2	14 × 32 = 448 bits
Instruction cache control registers: ICTRL	32 bits

Heavy ion experiments were performed at the CYCLONE cyclotron of HIF (Heavy Ion Facility), available at UCL (Université Catholique de Louvain-la-Neuve). A "high penetration particle cocktail" was used to give access to the tested device sensitive areas. As described in Table 7.6, this cocktail makes available a set of ions covering a large scope of energies. Heavy ion tests can be performed with the beam hitting the DUT at different incidence angles, thus allowing an increase in the resulting LET, which is inversely proportional to the cosine of the angle between the beam and the target board including the DUT. Heavy ion tests must be performed with beams having a penetration higher than the substrate's thickness after thinning (~70 μm).

Neon, Argon, and Krypton were the selected particles for the experiments presented in the following. Their use with the DUT board at 0° (normal incidence) and tilted at 45° incidence angles allowed to get cross-section measurements for LETs comprised between 3.3 and 47 MeV/mg/cm^2. It is important to note that the LET at the DUT active zone, generally called effective LET or LET$_{eff}$, will be lower than the one measured in the used facility, because incident beam particles lose energy while traveling across the substrate. This also applies in case of tilted beams where the particle trajectory is not perpendicular to the target circuit surface.

The POWER PC 7448 was exposed to those beams to get its static cross-section measurement, which, as said before, corresponds to the particle average number needed to provoke one SEE. We recall that the most significant SEE to be considered in the frame of this kind of experiments, are SEUs, SELs, and SEFIs.

SEU detection during operation under radiation is done by comparing the memory cells observed content to reference values. The SEFIs can be detected by the so-called watchdogs. Indeed, they return a timeout error whenever the application misses a check point. Finally, SEL detection is ensured by comparing the power consumption to rated values (with a 10% margin). In case a SEL is detected, the power supply is cut during some milliseconds, after which the DUT is reinitialized.

Table 7.6 Details of high penetration heavy ion cocktail available at CYCLONE

Ion	DUT energy (MeV)	Penetration (μm Si)	LET (MeV/mg/1312cm^2)
$^{13}C^{4+}$	131	266	1.2
$^{22}Ne^{7+}$	235	199	3.3
$^{28}Si^{8+}$	236	106	6.8
$^{40}Ar^{12+}$	372	119	10.1
$^{58}Ni^{17+}$	500	85	21.9
$^{83}Kr^{25+}$	756	92	32.4

The tested processors being manufactured with SOI technology are in principle immunes to SEL. In case of static tests, SEFIs are not counted. Indeed, even if SEFIs can be detected, it is not possible to identify the perturbed memory element as it can be provoked by corruption of many different memory elements' values, not accessible to users, through the instruction set.

Heavy Ion Tests Results

In Table 7.7 are provided the processor Power PC 7448 cache memory L1 and the registers static cross-sections (given in cm^2/bit). Test experiments were performed with a fluency of 5 × 105 particles.

Main conclusions about heavy ion tests are:

- The PC 7448 processor is immune to latchup (SEL). Immunity to SEL of the circuits manufactured in SOI technology is confirmed.
- Neither MCU nor MBU were observed.
- Sequence loss (timeout) faults were observed by means of a mechanism controlling the execution flow. In principle, it is not possible to identify the origin of such faults, which can be the consequence of a SEFI, a SEU in one of the registers monitoring the program sequence, or a bitflip in a processor's control unit memory cell.

The Power PC 7448 cross-sections, calculated from the average number of events observed in the processors issued from two different diffusion lots, are given in Figs. 7.5 and 7.6.

7.6.2.2 Neutron Testing

Neutron Test Conditions

Tested Devices

Neutron tests were performed on two Power PC 7447 and two Power PC 7448 devices, each one issued from two different diffusion lots.[3]

As neutrons have high penetration, thinning the DUTs was not necessary.

Table 7.7 Cross-sections, in cm^2/bit, of L1 cache memory and registers

Particle	Tilt (°)	Effective LET (MeV/mg/cm^2)	L1 cache	Registers
Ne	0	3.3	4.07E−09	3.12E−09
Ne	45	4.6	6.09E−09	9.09E−09
Ar	0	10	5.11E−09	1.30E−08
Ar	45	14	6.88E−09	1.87E−08
Kr	0	33	5.51E−09	1.73E−08
Kr	45	47	3.54E−09	8.10E−09

[3]Manufacturer is e2v semiconductors.

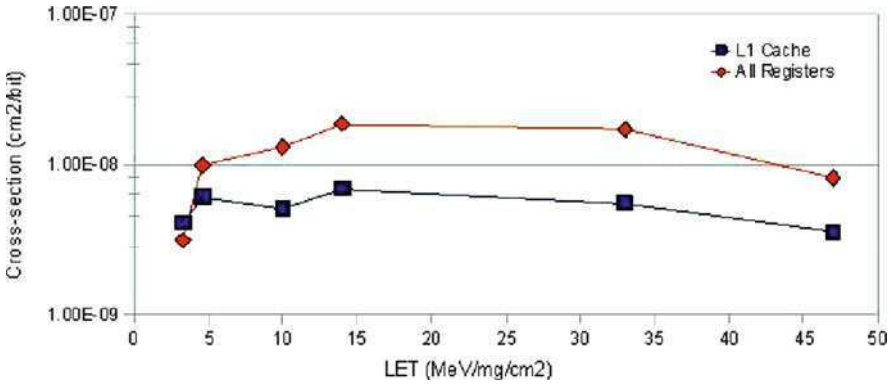


Fig. 7.5 Registers and L1 cache memory cross-section curves (in cm^2/bit)

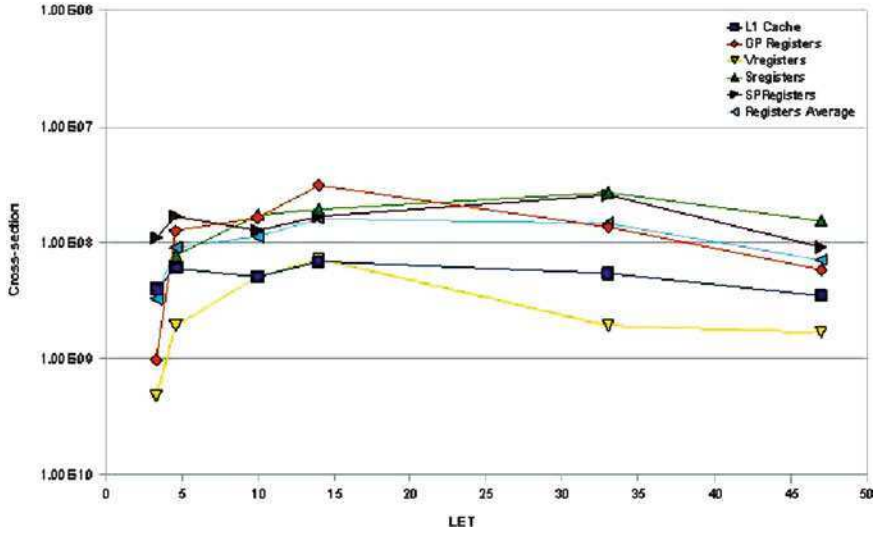


Fig. 7.6 Power PC 7448 accessible resources cross-sections curves (in cm^2/bit)

Test Conditions

In Table 7.8 are summarized test conditions adopted for neutron test experiments. The test strategy was the same as the one adopted for heavy ion testing.

The used static test program targets the Power PC accessible sensitive area: Register file and L1 cache memory, which represents a total of 8,929 bits for registers and 262,144 bits for cache memory.

Table 7.8 Neutron test conditions

	PC 7447A	PC 7448
Number of tested parts	Two parts issued from the same diffusion lot	Two parts issued from the same diffusion lot
Core power supply	1 V	1 V
I/Os power supply	2.5 V	2.5 V
Temperature	Room temperature	Room temperature
Type of test	Static	Static
Test pattern	CHB, CHBn, All0, All1	CHB, CHBn, All0, All1
Orientation	0 and 45°	0 and 45°
Operating frequency	33 MHz System Clock, 18× for the multiplier, ~600 MHz for the core	33 MHz System Clock, 18× for the multiplier, ~600 MHz for the core

The same set of registers was tested as the ones considered for heavy ions test (Section "Heavy Ion Tests Conditions"):

Neutron tests were performed at LANSCE (Los Alamos Neutron Science Centre), a facility which proposes a very powerful linear accelerator, capable of accelerating protons at 84% of the light's speed. Neutrons are produced when these protons collide with a tungsten target. These neutrons are used for a large scope of experiments; among them, the static cross-section of micro- and nano-electronic device measurements has a major place. The experiment's goal performed at LANSCE was to obtain the Power PC 7447A and 7448 cross-section measurement, from which the error rates in the atmosphere will be derived and evaluated as a function of the operation's number of hours. Such evaluation is generally performed in terms of FIT (Failure In Time), 1 FIT being equal to 1 error every 10^9 h of operation.

Neutron Test Results

Tables 7.9 and 7.10 report the measured cross-sections for the L1 cache and registers of the Power PC 7447A and 7448 processors. In these tables are reported different types of single events that can be observed during a radiation ground testing experiment: SBU (Single Bit Upset), MCU (Multiple Cell Upset), and SEL. When no errors are observed, during a particular experiment, the JEDEC JESD89A standard advises to calculate the cross-section considering an upper confidence level of 95%, which corresponds to a 3.7 errors upper limit. Used neutron beam was a high energy neutron with a spectrum close to the one at ground level, and a flux of 1.7×10^5 neutrons $\times$ cm^{-2}/s.

In Table 7.9, the rated cross-section is issued from the error rate observed while the DUT operates under rated conditions. The global cross-section corresponds to the cross-section average observed when performing tests under the following conditions: V_{min}, V_{nom}, and V_{max} using the checkboard (CHB) test pattern and $V_{nominal}$ with the "all to 0" and "all to 1" (All0 and All 1) test patterns.

Table 7.9 PC 7447A L1 cache and registers neutron cross-sections, in cm^2/bit

DUT	Test condition	Flips cross-section	SBU cross-section	MCU cross-section	SEL cross-section
PC 7447A L1 Cache	Rated	1.25E−14	1.25E−14	0 (UL = 3E−16)	0 (UL = 3E−16)
	Global	1.05E−14	1.05E−14	0 (UL = 6.5E−17)	0 (UL = 6.5E−17)
PC 7447A registers	Global	0 (upper limit = 1.1E−14)			

Table 7.10 PC 7448 L1 cache and registers neutron cross-sections, in cm^2/bit

DUT	Conditions du test	Flips cross-section	SBU cross-section	MCU cross-section	SEL cross-section
PC 7448 Cache L1	Nominal	1.14E−14	1.14E−14	0 (UL = 5E−16)	0 (UL = 5E−16)
	Global	1.17E−14	1.17E−14	0 (UL = 8E−17)	0 (UL = 8E−17)
PC 7448 registers	Global	0 (UL = 1.35E−14)			

It is important to notice that no errors were observed for the two tested processor registers, despite 4 full days' test duration. This is due to the fact that the registers' sensitive area (magnitude directly related with the total bit number) is too weak compared to the one to the cache memory. This entails a very long test under neutron beam, and so a very high cost to obtain a statistically representative number of errors. Absence of events on registers also shows that those blocks have a very low neutron sensitivity.

During neutron tests, the PC 7447A and 7448 were exposed to the neutron beam at room temperature, while they executed a checkboard algorithm. Results given in Tables 7.9 and 7.10 show the absence of latchup (SEL) events and MCU events. As the "Flip cross-section" is the sum of the "SBU cross-section" and the "MCU cross-section," this explains the cross-section values corresponding to the columns entitled "Flip cross-section" and "SBU cross-section" are the same. Indeed, these values will differ when MCUs are detected in case a single particle is perturbing more than one memory cell.

In Figs. 7.7 and 7.8 are given details about FIT vs test conditions (power supply) in the case where tested processors execute the Checkboard algorithm and consider the neutron flux in New York city (13 neutrons/cm^2/h). In these curves, the FIT is given per Megabit.

Neutron Tests Conclusions

Measurements issued from neutron test experiments are in the same order of magnitude than those obtained from other processors manufactured with SOI technologies. The absence of events on registers is justified by both the weak sensitivity and the total number of register bits, which is low compared to the tested L1 cache memory (about 8 Kbits vs. 256 Kbits) ones. Getting their cross-section under neutrons will thus require an unaffordable number of beam hours.

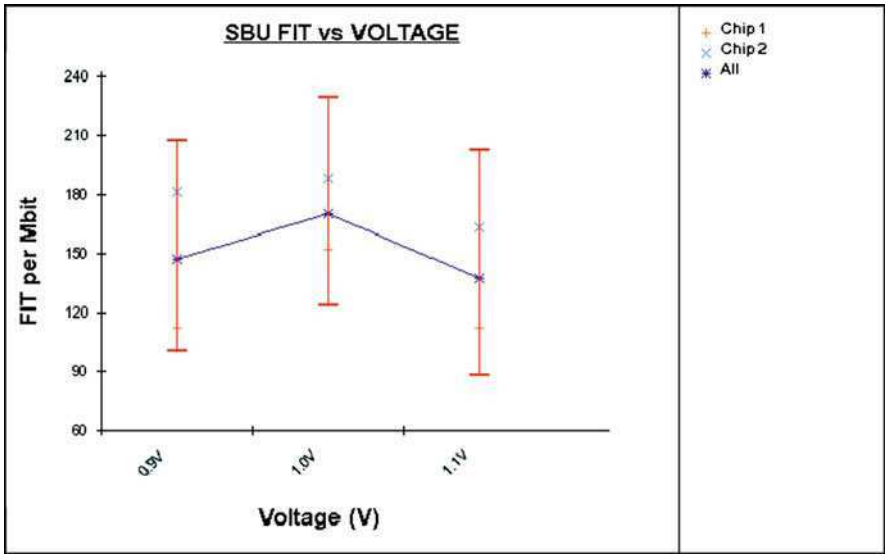


Fig. 7.7 FIT per Mbit of PC 7447A for different values of power supply and considering the New York City neutron flux

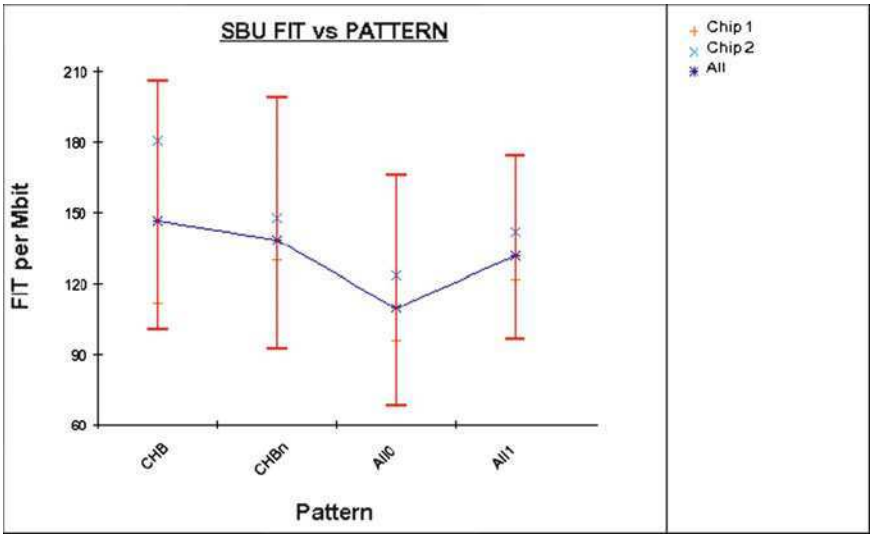


Fig. 7.8 FIT per Mbit of PC 7448 for different algorithms and considering the New York City neutron flux

It is important to notice that those measures are given with a 95% confidence interval according to the JEDEC standard. Considering the absence of multiple bit events (MBU), protection like ECC (Error Correcting Codes) can be adopted in case the measured FIT is not acceptable.

7.6.3 SRAM-Based Field Programmable Gate Arrays (FPGAs)

7.6.3.1 SRAM-Based FPGA Sensitive Areas

FPGAs use a logic gate grid similar to standard gate array, but programming is done by the customer, not by the manufacturer. FPGAs are usually programmed after being soldered down to the circuit board. In larger FPGAs, the configuration is volatile and must be re-loaded into the device whenever power is applied, or a different functionality is required. In this case, the configuration is typically stored in a SRAM memory. This memory is physically spread all over the die so that each configuration bit is placed next to the resource it configures.

Although those devices are convenient for developers, they are not well suited for applications devoted to operate in harsh environments. Indeed, some of the embedded user logic resources (RAM, flip-flops, Look-Up Tables, etc.) are sensitive to radiation. Moreover, the SRAM-based configuration memory is also sensitive because it is built with standard SRAM cells. The big threat of configuration memory is that a SEU in those cells may provoke a fault, which will remain until the component is re-configured. At the opposite, a fault in the user's memory will be temporary, an application reset is sufficient to erase it. So, when testing such a component, special care must be brought to the configuration memory as it is several times bigger than the sum of user's memory resources. Thus, the SEU occurrence probability in this memory is significantly higher than in any other memory cells.

Configuration is done by providing the FPGA an appropriate bitstream generated by the manufacturer's development tool software. This bitstream shall be stored in an external non-volatile memory. It is important to note that for use in harsh environment, this memory must be immune to radiation; otherwise, the FPGA will not be configured properly in case upset faults occur. Moreover, the readback function embedded in the FPGA allows reading the configuration memory content once it is configured. During radiation ground testing, this function allows the detection of faults in the configuration memory by comparing the bitstream after irradiation to a reference bitstream usually called *golden bitstream*. It is then possible to know the FPGA impacted resources by decoding the configuration bitstream obtained after radiation exposure. It is important to note than not all manufacturers provide full and detailed information for decoding the bitstreams.

7.6.3.2 Testing Procedures With Macro-Beams

Macro-beams such as heavy ions, protons, neutrons, etc., allow irradiating the whole component die at the same time. It is thus possible to obtain the whole component sensitivity within a few hours of test. Considering used fluxes are quite high (between hundred and thousand particles per cm^2/s) from such experiments, it is not possible to get the relationship between the upset location and its consequences at the application level. Indeed, the user has no information about the

resource mapping in the SRAM configuration memory. The only way to deal with this challenge is to use either heavy-ion micro-beams or laser beams.

Static Test

A static test is done in three phases. First, the FPGA must be configured without being exposed to particle fluxes. Then, the component is left idle (being only powered) under radiation for a certain time. This amount of time depends on the number of upsets generated per second. Indeed, the probability of having several upsets on the same bit, and thus not detecting them, increases drastically with the upset number. Finally, the beam is stopped to retrieve the configuration memory in order to compare it to the golden bitstream (Fig. 7.9).

Examples of results obtained for two SRAM-based FPGAs are given in Figs. 7.10 and 7.11. The first figure depicts the SEL cross-section vs LET of an Altera Stratix-II [12], while the second one depicts the SEU cross-section vs LET of a Xilinx Virtex-II [13].

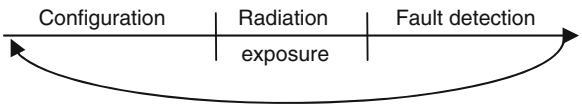


Fig. 7.9 SRAM-based FPGA static test procedure

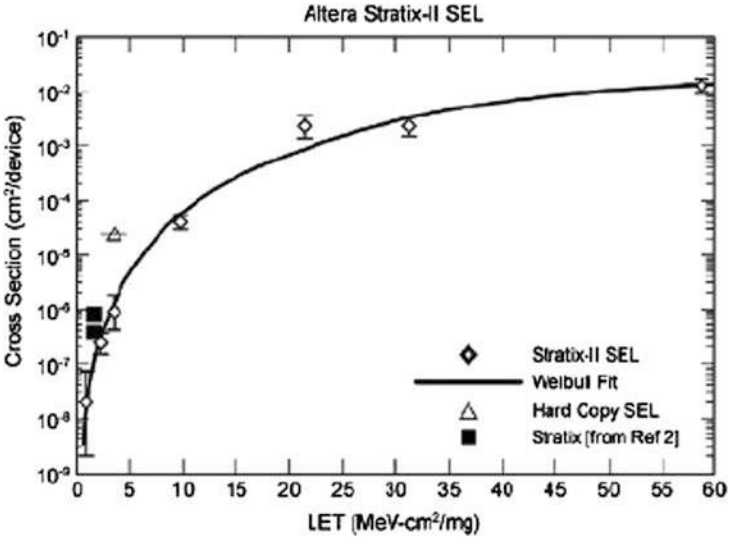


Fig. 7.10 Altera Stratix-II SEL cross-section vs. LET

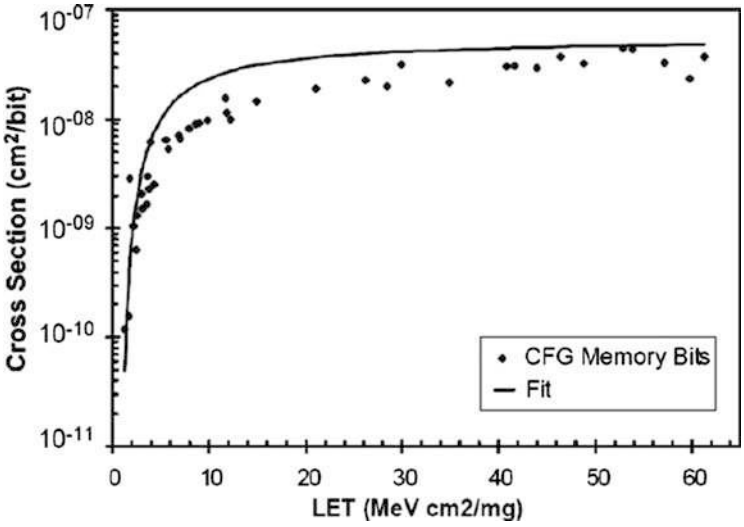


Fig. 7.11 SEU cross-section vs. effective LET for Virtex-II configuration memory cells

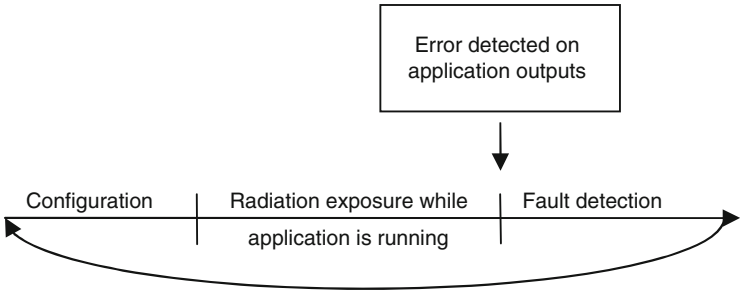


Fig. 7.12 FPGA dynamic test procedure

Dynamic Test

A dynamic test is basically done in the same way as the static one. It also has three phases: Configuration, exposure to radiation, and error detection. The first difference is the fact that during radiation exposure, the FPGA is running the target application instead of being idle. The second difference is that the application outputs are monitored and when an error is detected, it will end the test. Similar to that in the static test, the configuration memory is then read to detect faults (Fig. 7.12).

7.6.3.3 Laser-Beam Test Procedure

Laser beams can be used to get a very accurate mapping of the sensitive areas of the component under test. Their single shot ability allows injecting a fault at any

application clock cycle. This type of experiment is in general devoted to test a design critical function.

Static Test

A static test can be done in two ways, depending on the desired results type. The FPGA memory configuration can be retrieved, either after each shoot, or after having scanned a complete area (Fig. 7.13).

Dynamic Test

A dynamic test consists in injecting a fault at the desired chip location and clock cycle. A full geographical and temporal SEU mapping of the application is possible, but can be very time consuming. As an example, assuming a cycle (configuration-application run-memory readback-fault detection) takes around 1 s, the area to be mapped could be 1 mm^2, and the laser step is 1 μm; thus, the exhaustive scan would take almost 12 days. This corresponds to the scan for one application clock cycle, and must be repeated for all the remaining clock cycles (Fig. 7.14).

In conclusion, SRAM-based FPGAs are complex integrated circuits, and their current architecture makes them sensitive to natural radiation effects. However, available test platforms allow to predict the application behavior in radiation environments. Macro-beams provide the sensitivity of the whole device/application couple whereas laser beams are able to give exhaustive mapping of potential critical areas of an application. It is important to note that FPGA manufacturers do not follow the same confidentiality policy concerning the information they provide on their devices. Thus, deep analysis will be difficult for some parts.

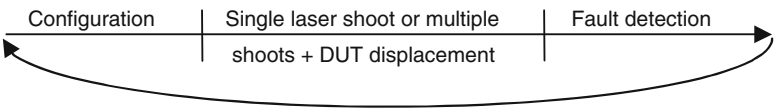


Fig. 7.13 FPGA static test using a laser beam procedure

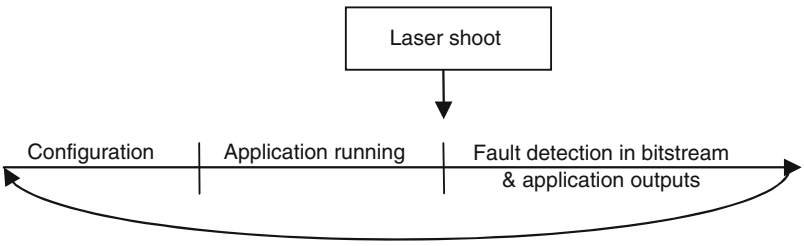


Fig. 7.14 FPGA dynamic test procedure

A new emergent technology based on flash memory cells, to replace their SRAM counterparts, might give good results in increasing the configuration memory robustness to radiation.

7.7 HW/SW Fault Injection Method for the Prediction of Dynamic Cross-Sections of Processor-Based Architectures: A Case Study

The dynamic cross-section, providing the particles average number required to get an error on an integrated circuit or system application outputs, is the final goal of the error-rate evaluation approach. Such a measure can be obtained from accelerated tests, but the whole process should be repeated each time the application is modified, and thus implies consequent development cost. HW/SW fault injection was proved being a suitable method to get dynamic cross-sections without performing radiation ground testing, and so it constitutes a good alternative dealing with the significant cost/effort required by such tests.

Fault injection can be applied at several levels. Software level fault-injection requires suitable target circuit models (SPICE, HDL). Such models allow having access to all the DUT potential target resources, which are often kept confidential by manufacturers. Moreover, the full software fault injection method's main drawback is the time required for its completion. Indeed, for a complex architecture, several weeks may be necessary to achieve the desired number of events. On the other hand, fault injection performed on a physical circuit can be run at the target application with full speed, thus significantly improving performances. In this case, the drawback is the limited access to some of potential target resources.

As most architectures are organized around processor-like devices, a generic strategy so-called CEU (Code Emulated Upset), which allows to estimate accurately the sensitivity to SEUs of any application running on such architecture, will be detailed in the following. This method, presented for the first time in [14], is described and then applied as a case study to a Power PC complex processor.

7.7.1 Error-Rate Prediction for an Application, a Case Study: The PPC7448

The expected result, when estimating an application error rate, is the average number of particles needed to get an observable application erroneous behavior. The application error rate can be calculated as follows from a fault injection experiment:

$$T_{\text{inj}} = \frac{\text{Number of errors}}{\text{Number of upsets}}.$$

Considering the static cross-section gives the particles average number needed to provoke one SEU in the target circuit; static cross-section multiplication by

error rate issued from fault-injection experiments allows predicting an application dynamic cross-section. Indeed, the dynamic cross-section, called σ_{SEU} in the following, corresponds to the particle's average number needed to provoke one erroneous result in the considered application. It is clear from the previous equation that σ_{SEU} can be predicted from the product of the error rate issued from fault injection experiments and the static cross-section:

$$T_{SEU} = \sigma_{static} \times T_{inj} = \frac{\text{Number of errors}}{\text{Number of particles}}.$$

The key point of such an approach is the simulation of upsets occurring during the execution of the considered application, a step that can be realized in different ways depending on the available model of the target processor. In cases where only a hardware version and the corresponding datasheet are available, the CEU method offers a good alternative to simulate errors induced on an application executed by a processor, resulting from the impact of energetic particles.

The CEU approach is based on the activation, at randomly chosen instants, of asynchronous interrupt signals which are available in most processor-like devices. After saving the context, the execution routine associated with the selected interrupt signal allows the modification of the content of a bit selected among those accessible through the processor's instruction set, thus simulating a SEU occurrence. The limitations of such a technique are related with the injection instant accuracy and the impossibility to access the whole set of target memory cells. Indeed, a SEU may occur at any instant, while interrupt signals are asserted once the instruction being executed is completed.

The CEU fault injection mechanism, illustrated in Fig. 7.15, is applied through the following steps:

- Processor Reset.
- Start program execution.
- Interrupt signal activation at a selected instant.
- Processor interrupts the main program and branches to the address where the interrupt routine is stored.
- Interrupt routine execution provokes the selected memory cell bit-flip.
- Return to the main program.
- Resume the main program execution with a bit having a wrong value.
- End of execution. Compare the provided results with the expected ones.

Unlike radiation ground test experiments, whose cost forces shortening their duration as much as possible and to stop them as soon as a "reasonable" number of events are detected, the CEU method can be applied without time constraints and thus allows injecting a very large number of SEUs. As an example, injection of 150,000 SEUs in PPC 7448 register bits required 2 days with the THESIC platform. Obtaining the same number of events with a particle accelerator would take around 4 complete days, resulting in a very high cost (around 600 Euros/h).

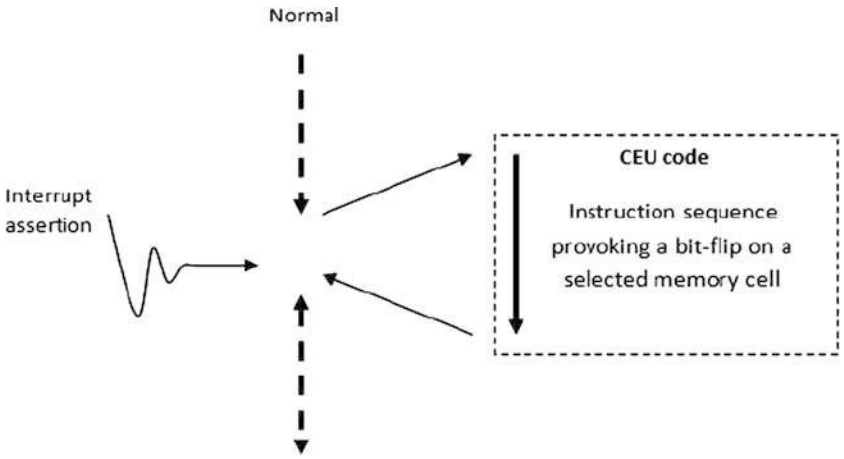


Fig. 7.15 Fault injection mechanism using interruption assertions

Three different types of situations were observed in the PPC 7448 as consequence of the injected SEUs:

- SEU has no consequence.
- SEFI: Program sequence loss, or exception assertion.
- SEU provokes an error in output data calculated by the program.

7.7.1.1 Implementing a CEU Approach for the POWER PC 7448 Processor

THESIC + Daughterboard for Power PC

A daughterboard allowing pigging back Power PC processors on the THESIC + tester [10] was developed in a way such that it can host either a PC 7447A or a PC 7448. The tester's architecture allows significant time and effort reduction to develop the required HW/SW when building a new daughterboard. Indeed, the design is limited to routing the processor's I/O to the motherboard's connector. Daughterboard also includes core and processor's I/O voltage regulators. Figure 7.16 shows the platform used to qualify the target Power PC processors under heavy ions. Those tests are performed in a vacuum chamber and the processor's thermal dissipation is achieved through a copper cover connected to the board's ground plane.

Software Developments

It is important to remind that the resources needed to provide tested processors with a suitable analog/digital environment are in principle all present in the THESIC's motherboard. Thus, using THESIC for a new processor test only requires a memory

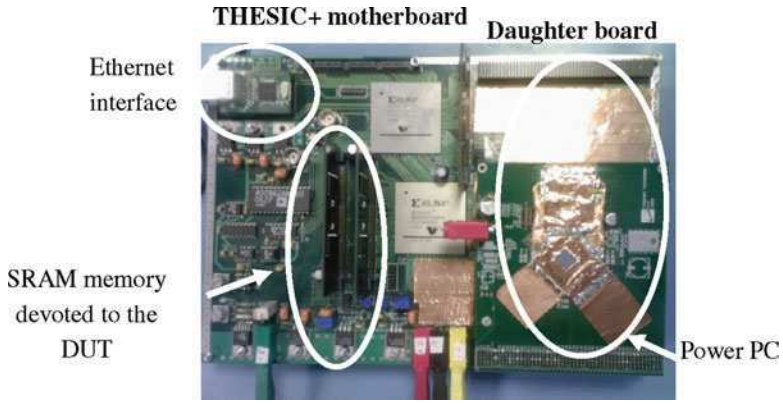


Fig. 7.16 Power PC's daughterboard plugged on THESIC + tester

controller implementation in the Chipset FPGA, this is to allow the tested processor to access its program code and data, both stored in THESIC's embedded SRAM.

In the Power PC processor's case, such a memory controller was written in Verilog HDL (Hardware Description Language). A control flow checker allowing different verification phases in the test program execution was also implemented in the THESIC Chipset FPGA.

A computer executing a program written in C language is used to control the Power PC during the experiment. It allows the Power PC to start the benchmark program execution and to check the generated data while being exposed to radiation particle flux.

Memory Controller

A memory controller module must be developed to suitably interface the DUT to the test platform. Such a module must take in charge the memory access cycle decoding at the address bus level. It must also share the access to the SRAM memory between the Power PC and the computer.

Execution Flow Control

For the two test strategies (static and dynamic), the test program executed by the target processor during its exposure to a particle beam, or during fault-injection sessions performed by HW/SW means, can be organized in many phases, each of them being associated with a counter implemented in the FPGA testbed. They are as follows:

- Booting: The tested processor configuration.
- Writing: Initialization with predetermined patterns the memory elements that must be tested.

- Waiting: Power PC processor exposure time to the particle beam during a static test.
- Dumping: Writing results in the tester memory.

Figure 7.17 illustrates the different steps in a static test execution. PIO[n] signals were implemented in the FPGA and are decoded when the processor accesses to a particular memory area. In case of a dynamic test, which is a typical application case execution, waiting and writing phases correspond to the calculation time needed to execute the application under test.

At the program execution end, each counter's content is compared to reference values which are obtained during the program execution without exposure to radiation. If these values are different from the expected one, a *sequence-loss* error will be detected. In that case, results associated with this execution are not exploited to avoid data misinterpretation problems. THESIC's architecture flexibility makes it possible to control a shutter (physical device allowing to stop the particles before they reach the DUT) in order to adjust the beam fluency; a step which is mandatory to guarantee the measured error rates validity. It is important to note that boards used to perform complex devices radiation ground testing, are generally based on commercially available boards developed around the tested device, and do not offer the possibility of implementing such an adjustment.

The developed control flow execution also allows the accurate measure of the exposure time to radiation beam, and thus to derive the exact particle fluency (flux integrated in time) to which the studied application was exposed.

The exposure time for n executions is:

$$E = \sum_{i=0}^{n} \text{running}_i.$$

The running counter in the THESIC's memory covers result initialization time, waiting time and writing time. To guarantee the accuracy of these estimations,

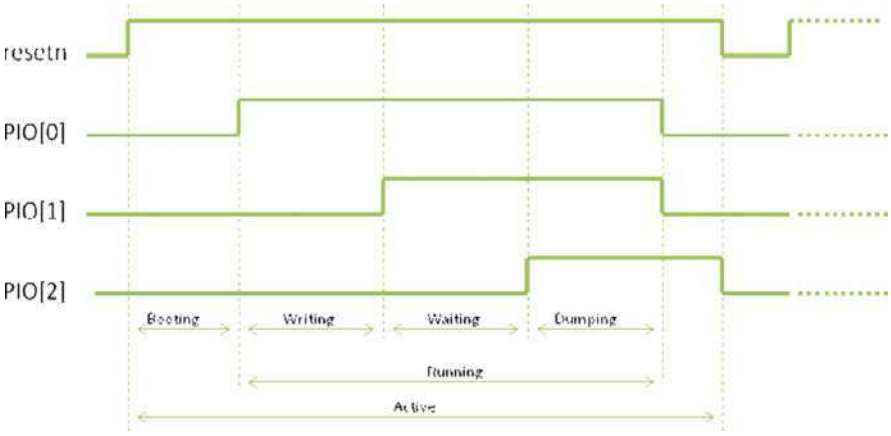


Fig. 7.17 Tested processor's execution flow

initialization and writing times must be negligible with respect to beam exposure time.

The real fluency seen by the tested processor is given by:

$$F_R = \frac{F}{T} \times E,$$

where F is the fluency (flux integrated in the test time) provided by the beam facility, while T is the total exposure time during the performed experiment. The average flux (number of particles per units of surface and time) is adjusted by coefficient E, in order to retrieve the number of particles to which the DUT was exposed during the application's execution.

Dynamic Cross-Section Measures

As stated before, in programs used to evaluate static cross-sections, the tested processor does not perform calculations; the goal being to get the sensitivity measurements to heavy ion effects for different memory areas accessible through the instruction set. Results about the studied processor's sensitivity were obtained by performing radiation ground testing with the same particle beams used for static tests. A realistic program, called SAC,[4] devoted to the orbit calculation was executed during this campaign. SAC aims at providing a stratospheric balloon gondola stabilization, having as payload a telescope for the Cold Universe observation within the submillimetric domain (wavelength comprised between 200 μm and 1.2 mm). The stabilization is ensured by a controller based on a Kalman filter, taking into account data issued from different measures provided by sensors (inertial computer, stellar sensor, and GPS data).

From the SAC program, an executable code executed in the THESIC platform was obtained to get reference results. A set of data emulating the sensors' measures was used during the SCA program execution.

During the heavy ion tests performed in Louvain-la-Neuve HIF cyclotron, the Power PC 7448 processor was exposed to heavy ion beams during the SCA program execution (in a looped way). Dynamic cross-sections were measured for two different configurations of the tested Power PC: L1 cache memory activated and L1 cache memory deactivated.

Experimental Results

Table 7.11 summarizes the experimental results obtained as a consequence of 150.000 SEUs injected using THESIC+.

[4]That program, provided by CNES, is issued from the software "SCA TAFT/PRONAOS". This software is a System of Attitude Control (SAC) developed for CNES in the PRONAOS (*PROjet National d'AstrOnomie Submillimétrique*) project frame.

In Tables 7.12 and 7.13 are confronted measures and error-rate predictions issued from fault injections performed following the CEU approach on the PPC 7448 for the two cache memory configurations.

These results clearly show the pertinence of the adopted method to predict application error-rates, even when executed by advanced processors having complex architectures including cache memories and parallel execution of instructions.

Error-Rate Prediction from Different Orbits

Table 7.14 summarizes the SEU error-rate results in orbit for the L1 cache and the registers of the PC 7448 due to heavy ions, calculated[5] for GEO, Polar and ISS (International Space Station) orbits. These calculations were performed with the software OMERE[6] using the CREME86 model with $M = 3$ configuration (90% worst case).

It is important to note that the "TOTAL" lines do not take into account the L2 cache of 8 Mbits, which was not considered during radiation ground test experiments.

From these results, it is clear that the PC 7448 processor presents a weak error rate for the different considered orbits:

- GEO orbit: around 1 error per month
- Polar orbit: about 1 error every 2 months
- ISS orbit: around 1 error every 2 years

Table 7.11 Error rates obtained as the consequence of 150,000 SEUs injected using THESIC+

Type	
Program sequence loss	2.2%
Exception	0.9%
Error observed	80%
Corrected results	16.9%

Table 7.12 Predictions vs. measures (cache memory is disabled)

Ions	Predicted σ_{SEU}	Measured σ_{SEU}
Argon	1.96E−06	1.84E−06
Krypton	3.82E−06	3.56E−06

Table 7.13 Predictions vs. measures (cache memory enabled)

Ions	Predicted σ_{SEU}	Measured σ_{SEU}
Argon	2,12E−05	2.04E−05
Krypton	3.24E−05	3.17E−05

[5]By the French Space Agency (CNES).

[6]Version 3.1.2.0 by TRAD-CNES.

Table 7.14 PC 7448 Predicted error rate for different orbits

	Number of bits	Errors/bit/day	Errors/device/day	Orbits
L1 cache	589,824	5.50E−08	3.24E−02	GEO (35,870 km)
GP registers	1,024	7.05E−08	7.22E−05	
V registers	4,096	1.30E−08	5.32E−05	
S registers	512	4.41E−08	2.26E−05	
SP registers	3,264	2.37E−07	7.74E−04	
Total	598,720	4.20E−07	3.34E−02	
L1 cache	589,824	2.56E−08	1.51E−02	LEO1 Pol (98°, 800 km, 800 km)
GP registers	1,024	3.20E−08	3.28E−05	
V registers	4,096	5.14E−09	2.11E−05	
S registers	512	1.17E−08	5.99E−06	
SP registers	3,264	1.36E−07	4.44E−04	
Total	598,720	2.10E−07	1.56E−02	
L1 cache	589,824	2.47E−09	1.46E−03	LEO2 ISS (51.5°, 400 km, 400 km)
GP registers	1,024	4.79E−09	4.90E−06	
V registers	4,096	6.77E−10	2.77E−06	
S registers	512	1.46E−09	7.48E−07	
SP registers	3,264	2.20E−08	7.18E−05	
Total	598,720	3.14E−08	1.54E−03	

Heavy Ion Test Conclusions

From the presented results, it may be concluded that the PC 7448 processor is not sensitive to latchups events, and has a weak sensitivity to upsets (except the L2 cache which was not tested), and thus it can be a suitable candidate for space applications needing a huge calculation power. Nevertheless, it is not advised to use it for critical functions or in functions which can integrate hardware and/or software mechanisms to detect, or detect and correct errors. As an example of suitable applications for the PC 7448, it can be mentioned intensive data processing in satellite's payload.

7.8 Conclusions

This chapter dealt with the application of an approach to complex integrated devices to predict error rates due to soft-errors induced by ionizing radiation. It combines the intrinsic sensitivity to soft errors of the tested device issued from radiation ground testing performed with suitable particle accelerator facilities with results issued from soft-errors fault injection experiments performed by hardware/software means by using suitable test platforms.

Such an approach allows in dealing with bit flips that occur in any memory cell of an advanced processor as long that they are accessible through the instruction set. In such devices, the huge sensitive area corresponding to register files, cache memories, and any other memory cells accessible by means of the instruction set makes the predictions based on the presented approach particularly accurate.

The proposed approach was applied to the Power PC 7448 processor, one of the most recent commercial 32-bits processors having a superscalar RISC architecture. Obtained results proved the validity of the application error-rate prediction validity. Those predictions were proved to be very close from error rates measured in a cyclotron facility, while the studied processor was executing the selected application: a flight software devoted to a satellite orbital control.

A current trend to get an objective feedback about advanced integrated devices sensitivity to the effects of energetic particles present in the Earth's atmosphere, is to perform real life experiments in which large sets of target devices are monitored for a long period of time (months, years, etc.). This gives a chance to get potential statistics about error rates induced by atmospheric neutrons. In order to decrease the time required to get significant results, such experiments can either be installed in high altitude sites, or be embedded in stratospheric balloon payloads, or even installed in deep underground sites in order to eliminate SEUs due to alpha particles and only keep the ones induced by neutrons. A representative experiment is the Rosetta project [15, 16], which deals with different generation of SRAM-based FPGAs.

Acknowledgments The authors would like to thank Robert Ecoffet and Michel Pignol from CNES for their significant contribution to these researches started in 1989 and Guy Berger, responsible of the HIF cyclotron where experiments were conducted, who offered a continuous support to this work.
They thank also Dominique Bellin, from e2v, and IROC Company who significantly contributed to the application of the test methods and tools developed at TIMA for the Power PC processor in the frame of the SCADRI project if Rhône-Alpes aeronautics & space Cluster.
Finally, the authors thank Joseph Foucard for his help in the redaction of this chapter.

References

1. E. Normand, "Single-event effects in avionics", IEEE Transactions on Nuclear Science, Vol. 43, No 2, pp. 461–474, Apr. 1966.
2. T. Ma, P. Dressendorfer, "Ionizing Radiation Effects in MOS Devices and Circuits", Wiley Eds., New York, 1989.
3. F. Faccio, "Design Hardening Methodologies for ASICs", Radiation Effects on Embedded Systems, ISBN-10 1-4020-5645-1, Springer, 2007.
4. R. Velazco, F. Faure, "Error Rate Prediction of Digital Architecture: Test Methodology and Tools", Radiation Effects on Embedded Systems, ISBN-10 1-4020-5645-1, Springer, 2007.
5. Bezerra F. et al, "Commercial Processor Single Event Tests", 1997 RADECS Conference Data Workshop Record, pp. 41–46.
6. R. Velazco, S. Karoui, T. Chapuis, D. Benezech, L.H. Rosier, "Heavy ion tests for the 68020 Microprocessor and the 68882 Coprocessor", IEEE Transactions on Nuclear Science, Vol. 39, No 3, Dec. 1992.
7. J.H. Elder, J. Osborn, W.A. Kolasinsky, R. Koga, "A method for characterizing microprocessor's vulnerability to SEU", Transactions on Nuclear Science, Vol. 35, No 6, pp. 1679–1681, Dec. 1988.
8. S. Duzellier, G. Berger, "Test Facilities for SEE and Dose Testing", Radiation Effects on Embedded Systems, ISBN-10 1-4020-5645-1, Springer, 2007.
9. JEDEC standard, "Measurement and Reporting of Alpha particles and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices", JESD89 Aug. 2001.

10. F. Faure, P. Peronnard, R. Velazco, “Thesic+: A flexible system for see testing”, In Proceedings of RADECS, 2002.
11. R. Velazco, P. Cheynet, A. Bofill, R. Ecoffet, “THESIC: A testbed suitable for the qualification of integrated circuits devoted to operate in harsh environment”, In Proceedings of IEEE European Test Workshop, pp. 89–90, 1998.
12. Gregory R. Allen, Gary M. Swift, “Single Event Effects Test Results for Advanced Field Programmable Gate Arrays”, IEEE Radiation Effects Data Workshop, July 2006.
13. C. Yui, G. Swift, C. Carmichael, “Single Event Upset Susceptibility Testing of the Xilinx Virtex II FPGA”, MAPLD, 2002.
14. R. Velazco, S. Rezgui, R. Ecoffet, “Predicting error rate for microprocessor-based digital architectures through C.E.U. (Code Emulating Upsets) injection”, IEEE Transactions on Nuclear Science, Vol. 47, No 6, part 3, Dec. 2000.
15. A. Lesea, S. Drimer, J. Fabula, C. Carmichael, P. Alfke, “The Rosetta experiment: Atmospheric soft error rate testing in differing technology FPGAs”, IEEE Transactions on Device and Materials Reliability, Vol. 5, No 3, September 2005.
16. A. Lesea, “Continuing Experiments of Atmospheric Neutron Effects on Deep Submicron Integrated Circuits”, Xilinx WP286, March 10, 2008.

Chapter 8
Circuit-Level Soft-Error Mitigation

Michael Nicolaidis

In nanometric technologies, circuits are increasingly sensitive to various kinds of perturbations. Soft errors, a concern in the past for space applications, became a reliability issue at ground level. Alpha particles and atmospheric neutrons induce single-event upsets (SEUs) affecting memory cells, latches, and flip-flops, and single-event transients (SETs) initiated in the combinational logic and captured by the associated latches and flip-flops. To face this challenge, a designer must dispose a variety of soft-error mitigation schemes adapted to various circuit structures, design architectures, and design constraints. In this chapter, we describe several SEU and SET mitigation schemes that could help designers to meet their reliability constraints.

8.1 Introduction

In the past, progress in VLSI technologies improved dramatically the reliability of electronic components, restricting the use of fault tolerance in a small number of application domains. We recently reached a point where these trends are reversed. Drastic device shrinking, power supply reduction, and increasing operating speeds that accompanied the technological evolution to nanometric technologies have drastically reduced the reliability of deep submicron ICs. A significant problem is related to soft errors induced, on the one hand, by alpha particles produced by radioactive isotope traces found in packaging, bonding, and die materials and, on the other hand, by atmospheric neutrons created by the interaction of cosmic rays with the atmosphere [1]. While alpha particles are electrically charged and create

M. Nicolaidis
TIMA Laboratory (CNRS, Grenoble INP, UJF), Grenoble, France
e-mail: michael.nicolaidis@imag.fr

M. Nicolaidis (ed.), *Soft Errors in Modern Electronic Systems*,
Frontiers in Electronic Testing 41, DOI 10.1007/978-1-4419-6993-4_8,
© Springer Science+Business Media, LLC 2011

a track of electron–hole pairs when they pass through an IC, neutrons are electrically neutral and create soft errors in an indirect manner: energetic neutrons may interact with silicon, oxygen, or other atoms of the chip to create electrically charged secondary particles able to induce soft errors, as described in Chaps. 1 and 2. It is to note that the interaction can create numerous secondary particles that could simultaneously affect several circuit nodes. Multiple nodes can also be affected by a single particle, as technology shrinking reduces the size of cells and bring sensitive nodes closer to each other. Thermal neutrons may also represent a major source of soft errors, but this source is significantly reduced by BPSG removal in recent technologies [2].

When a sensitive node, typically the drain of an off transistor, is in the proximity of the ionization track of an electrically charged particle, it collects a significant part of carriers (holes or electrons), resulting in a transient current pulse on this node. The effect of this pulse depends on the type of the cell to which the stricken node belongs. In a storage cell (e.g., a memory cell, a latch, or a flip-flop), a sufficiently strong pulse will reverse the cell state, resulting in a single-event upset (SEU). This is the most typical case of soft errors. When the collecting node belongs to a logic gate, the transient current pulse is transformed to a voltage pulse (single-event transient, SET), whose characteristics depend on the characteristics of the current pulse and on the electrical characteristics of the collecting node (load, strength of the transistors driving the node). Then, if the duration of the voltage pulse is larger than the logic transition time of the subsequent gates, it can be propagated through one or more paths of the combinational logic to reach some latches or flip-flops. Such a path is composed of a succession of input–output pairs of a set of gates connected in series. The pulse propagation is conditioned by the state of the combinational logic. Indeed, a controlling value on one input of a gate (e.g., a 0 on a NAND input or a 1 on a NOR input) will block the propagation of an error from another input of the gate to the gate output. Thus, a transient pulse can be propagated only through a sensitized path, that is, through a path such that all the inputs of the gates traversed by the path, excepting their inputs belonging to the path have non-controlling values. Finally, when the pulse reaches a latch or a flip-flop, it will be captured to produce a soft error only if it overlaps with the clock event. These conditions make soft errors induced by SETs more rare than SEUs induced when a particle strikes a node of a storage cell. However, the situation is changing for two reasons, as the logic-transition time of logic gates becomes extremely short, the durations of transient pulses become larger than this time and they are no more filtered by the electrical characteristics of the gates [3–5]. In addition, as the clock frequencies increase, the probability that the transient pulse overlaps with the clock event also increases [3–5]. Finally, in many situations, the path sensitization condition may not have a significant impact on reducing circuit sensitivity to SEUs, since a node of a combinational circuit is usually connected to the circuit outputs through several paths. So, the probability that none of these paths is sensitized can be low. Also, pulse propagation through several paths may result on several pulses that reach several circuit outputs at different times. This increases

the probability that at least one of these pulses overlaps with the clock event when it reaches a latch/flip-flop.

The soft-error rate (SER) produced by these effects may exceed the failure in time (FIT) specifications in various application domains. In such applications, soft-error mitigation schemes should be employed for memories and eventually for logic, while the continuous technological scaling also introduces new reliability issues due to PVT (power-voltage-process) variability, increasing sensitivity to EMI (electromagnetic interferences), and accelerating aging. The necessity for nanometric technologies (below 100 nm) and for low supply voltages, to use error correcting codes (ECC) in memories, *hardened latches and (flip-flops* and *concurrent error detection and retry* in logic, in order to mitigate soft errors and other issues making devices to behave statistically, and the increasing needs for such solutions as we move below 30–50 nm, was predicted more than one decade ago [6–10].

Soft-error mitigation techniques include approaches using process modifications, such as for instance the removal of BPSG for mitigating soft errors produced by thermal neutrons [1, 2]; approaches adding extra process steps, such as for instance the addition of DRAM-type capacitance to the nodes of memory or latch cells [11]; and design-based approaches. In this chapter, we only address design solutions. These solutions include electrical-level, gate-level, and architectural-level approaches.

Error correcting codes is the conventional solution used to reduce SER in memories. It may incur a moderate area, power and speed penalties acceptable by most designers. But in some designs, the speed and/or area penalty incurred by ECC may be unacceptable. In addition, in some types of memories, ECC may be inapplicable or may incur very high cost. Duplication and comparison or triple modular redundancy (TMR) and majority voting are the most commonly used solutions for reducing SER induced by SEUs and SETs in logic parts. But they incur excessive area and power penalties, making them unacceptable in most designs.

In this paper, we discuss various alternative design solutions for soft-error mitigation, aimed to cope with some shortcomings of conventional solutions.

8.2 Design for Soft-Error Mitigation in Memories

Memories represent the largest parts of modern designs. In addition, they are more sensitive to ionizing particles than logic, since they are designed to reach the highest possible density. As a matter of fact, the SER of modern designs is dominated by the SER of their memories. Therefore, protecting memories is the first priority in designs requiring SER reduction. While permanent faults in memories can be fixed by various Built-In Self-Repair (BISR) techniques (see for instance [12]), this not possible for soft errors. For these faults, on-line fault masking approaches are needed.

8.2.1 Single-Error Correcting Double-Error Detecting Codes

Various error detecting and error correcting codes have been developed for memories. The most commonly used code is the Hamming code [101]. This code is able to correct a single error and also detect double errors if it is extended with an additional parity bit. To encode a set of information bits (to be also referred as data bits), this code uses r check bits, where r is the smaller integer verifying the relation $r \geq \lceil \log_2(k + r + 1) \rceil$, with k the number of information (data) bits and $n = k + r$ the total number of bits of the code. For instance, for 8 information bits, we need 4 check bits and for 16 information bits, we need 5 check bits. The code locates any single error by means of a syndrome that encodes the error position. For doing so, a word comporting information bits and check bits is created. In this word, each bit position j, such that j is a power of 2, is occupied by a check bit, and the other bit positions are occupied by the information bits. For instance, for 8 information bits (D_7, D_6, D_5, D_4, D_3, D_2, D_1, D_0), 4 check bits C_3, C_2, C_1, C_0 are added. The code words are formed as $(B_{12}, B_{11}, B_{10}, B_9, B_8, B_7, B_6, B_5, B_4, B_3, B_2, B_1) = (D_7, D_6, D_5, D_4, C_3, D_3, D_2, D_1, C_2, D_0, C_1, C_0)$. That is, C_0 occupies position B_{2^0}(i.e., B_1), C_1 occupies position B_{2^1} (i.e., B_2), C_2 occupies position B_{2^2} (i.e., B_4), and C_3 occupies position B_{2^3} (i.e., B_8).

The check bits are computed as follows: each bit position j in the code word corresponding to an information bit (i.e., j is not a power of 2) is expressed in binary code, that is, $j = \alpha_{r-1,j}2^{r-1} + \alpha_{r-2,j}2^{r-2} + \cdots + \alpha_{1,j}2^1 + \alpha_{0,j}2^0$, where the term $\alpha_{i,j}$ takes the value 0 or 1 (for instance, $9 = \alpha_{3,9}2^3 + \alpha_{2,9}2^2 + \alpha_{1,9}2^1 + \alpha_{0,9}2^0 = 1.2^3 + 0.2^2 + 0.2^1 + 1.2^0$, thus $\alpha_{3,9} = 1$, $\alpha_{2,9}2 = 0$, $\alpha_{1,9} = 0$, $\alpha_{0,9} = 1$). Then, the check bit $B_{2^i} = C_i$ is computed as the sum modulo 2 (XOR) of the information bits B_j for all j, such that the coefficient $\alpha_{i,j}$ in the above expressions is 1. That is,

$$B_{2^i} = C_i = \sum_{j=1, j \neq 2^q}^{k+r} a_{i,j} B_j, \quad \forall i \in \{0, 1, \ldots, r-1\}, \tag{8.1}$$

For instance, for the case of 8 information bits considered above, we find $C_3 = B_{2^3} = 1 \cdot B_{12} \oplus 1 \cdot B_{11} \oplus 1 \cdot B_{10} \oplus 1 \cdot B_9 \oplus 0 \cdot B_7 \oplus 0 \cdot B_6 \oplus 0 \cdot B_5$, $0 \cdot B_3 = B_{12} \oplus B_{11} \oplus B_{10} \oplus B_9$, since we find $\alpha_{3,12} = 1$, $\alpha_{3,11} = 1$, $\alpha_{3,10} = 1$, $\alpha_{3,9} = 1$, $\alpha_{3,7} = 1$, $\alpha_{3,6} = 0$, $\alpha_{3,5} = 0$, $\alpha_{3,3} = 0$, as the binary representations of 12, 11, 10, and 9 contain the term 2^3, while the binary representations of 7, 6, 5, and 3 do not contain it.

Similarly, we can compute the other check bits, resulting in the relationships:

$$\begin{aligned}
B_1 = C_0 &= B_3 \oplus B_5 \oplus B_7 \oplus B_9 \oplus B_{11} = D_0 \oplus D_1 \oplus D_3 \oplus D_4 \oplus D_6, \\
B_2 = C_1 &= B_3 \oplus B_6 \oplus B_7 \oplus B_{10} \oplus B_{11} = D_0 \oplus D_2 \oplus D_3 \oplus D_5 \oplus D_6, \\
B_4 = C_2 &= B_5 \oplus B_6 \oplus B_7 \oplus B_{12} = D_1 \oplus D_2 \oplus D_3 \oplus D_7, \\
B_8 = C_3 &= B_9 \oplus B_{10} \oplus B_{11} \oplus B_{12} = D_4 \oplus D_5 \oplus D_6 \oplus D_7.
\end{aligned} \tag{8.2}$$

Let $(B'_{r+k}, \ldots, B'_j, \ldots, B'_2, B'_1)$ be the word $(B_{r+k}, \ldots, B_j, \ldots, B_2, B_1)$ received after transmission or read after storage in a memory. This word may contain errors. The correction of errors in $(B'_{r+k}, \ldots, B'_j, \ldots, B'_2, B'_1)$ is performed by generating a syndrome comporting r bits. To compute this syndrome, we first use (8.1) to generate the check bits from the data bits B'_j (i.e., from the bits B'_j with $j \neq 2^i$), and then we bit-wise XOR each of the newly computed check bit B''_{2^i} with the corresponding bit B'_{2^i}. Thus, the bit S_i of the syndrome is computed as

$$S_i = B'_{2^i} \oplus \sum_{j=1, j\neq 2^q}^{k+r} a_{i,j} B'_j = \sum_{j=1}^{k+r} a_{i,j} B'_j, \quad \forall i \in \{0, 1, \ldots, r-1\}. \tag{8.3}$$

If there is a single error, then $B'_j = B_j$ for each bit B'_j, excepting for the bit B'_m affected by the error. For this bit, it will be $B'_m = B_m \oplus 1$. By using these relationships to substitute the bits B'_j in (8.3) and then by substituting the bits B_{2^i} from (8.1), we find $S_i = a_{i,m}$. As $a_{i,j}$ are the coefficients of the binary representation of position j, the relation $S_i = a_{i,m}$ shows that the r-bits syndrome indicates in binary code the faulty position m. Thus, the error is corrected by inverting the bit B'_m, which in this way recovers its correct value B_m. If there is no error in the bits B'_j (i.e., $B'_j = B_j$ for all j), then all bits S_i of the syndrome will be 0, indicating an error-free word.

Relationships (8.1) and (8.3) lead on a memory error correction architecture shown in Fig. 8.1. On the write side, the check bits are generated from the data to be written in the memory. Once the check bits are generated, both data and check bits are written in the memory. On the read side, new check bits are generated from the read data. The new check bits are bit-wise XORed (XOR1 block) with the read check bits, to compute the syndrome. The r-bit syndrome encoded in binary is mapped into an n-bit syndrome encoded in the 1-out-of-n code (1-hot code). This mapping is realized by the block referred in Fig. 8.1 as 1-hot (a decoder circuit similar to the address decoders of memories). In case of a single error, the unique 1 in the 1-hot syndrome indicates the erroneous bit position. Thus, the 1-hot syndrome is bit-wise XORed with the code word composed of the read data and the read check bits, to provide the corrected data and check bits on the output of the XOR2 block.

Relationships (8.1) and (8.3) can be expressed in a more compact form by means of parity matrixes. For computing the r Hamming check bits from k data bits, we

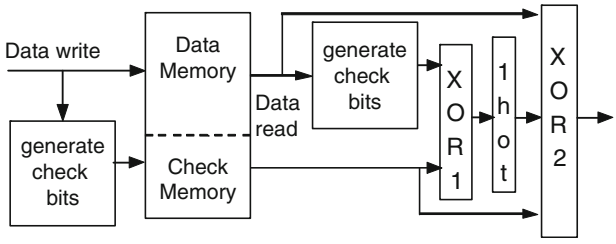


Fig. 8.1 Memory architecture for Hamming error correction

will use the relationship $C^T = \mathrm{Pc}\, D^T$, where C and D are the row matrixes of check bits and data bits $C = [C_{r-1}, \ldots, C_1, C_0]$, $D = [D_{k-1}, \ldots, D_1, D_0]$; C^T and D^T are the transpose of C and D; and Pc a $k \times r$ parity matrix, whose terms are the $k \cdot r$ coefficients $a_{i,j}$ used in (8.1). More precisely, from (8.1) we can find that at each data bit B_j ($j \neq 2^q$) corresponds a column of matrix Pc. The terms of this column are the r coefficients $a_{i,j}$ of the binary representation of j. All sums involved in the relationship $C^T = \mathrm{Pc}\, D^T$ are considered modulo 2. The parity matrix Pc for generating the Hamming check bits corresponding to 8 information bits can be obtained from (8.2):

$$\begin{array}{cccccccc} B_{12} & B_{11} & B_{10} & B_9 & B_7 & B_6 & B_5 & B_3 \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} & . \end{array}$$

For easier reference, we show on the top of each column the corresponding bit B_j. We easily check that the terms of column labeled by B_j are the coefficients $a_{i,j}$ of the binary representation of j.

For computing the r bits composing the Hamming syndrome from the k data bits and the r check bits, we will use the relationship $S^T = \mathrm{Ps}\, B^T$, with $S = [S_{r-1}, \ldots, S_1, S_0]$, $B = [B_{k+r}, \ldots, B_2, B_1]$, S^T and B^T the transpose of S and B, and Ps a $(k + r)xr$ parity matrix Ps (often referred in the literature as H matrix), whose terms are the $(k + r)r$ coefficients $a_{i,j}$ used in (8.3). All sums involved in the relationship $S^T = \mathrm{Ps}\, B^T$ are considered modulo 2. Matrix Ps is easily determined by adding three new columns in matrix Pc. Each of these columns is formed by the r coefficients $a_{i,2^q}$ of the binary representation of the position 2^q of check bit B_{2^q}. Obviously, each of these columns contains exactly one 1 (placed in the $q + 1$ row).

For $k = 8$, the parity matrix Ps (the H matrix) is obtained from the matrix Pc by adding $r = 4$ columns (shown in bold characters) corresponding to the coefficients of positions $j = 2^q$. The bit B_j is placed on the top of the corresponding column j for easier reference:

$$\begin{array}{cccccccccccc} B_{12} & B_{11} & B_{10} & B_9 & \mathbf{B_8} & B_7 & B_6 & B_5 & \mathbf{B_4} & B_3 & \mathbf{B_2} & \mathbf{B_1} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & \mathbf{0} & 1 & 0 & 1 & \mathbf{0} & 1 & \mathbf{0} & \mathbf{1} \\ 0 & 1 & 1 & 0 & \mathbf{0} & 1 & 1 & 0 & \mathbf{0} & 1 & \mathbf{1} & \mathbf{0} \\ 1 & 0 & 0 & 0 & \mathbf{0} & 1 & 1 & 1 & \mathbf{1} & 0 & \mathbf{0} & \mathbf{0} \\ 1 & 1 & 1 & 1 & \mathbf{1} & 0 & 0 & 0 & \mathbf{0} & 0 & \mathbf{0} & \mathbf{0} \end{pmatrix} & . \end{array}$$

8.2.1.1 Detection of Double Errors

If there is a double error, then $B'_j = B_j$ for each bit B'_j, excepting for two bits B'_p and B'_q of two different positions p and q affected by the error. For these bits, it will be

$B'_p = B_p \oplus 1$, $B'_q = B_q \oplus 1$. By using these relationships to substitute the bits B'_j in (8.3) and then by substituting the bits B_{2^i} from (8.1), we find $S_i = a_{i,q} \oplus a_{i,r}$. Since $q \neq p$ and $a_{i,q}$ are the coefficients of the binary representation of q and $a_{i,p}$ are the coefficients of the binary representation of p, $a_{i,q}$ and $a_{i,p}$ cannot be equal for all i. Thus, S'_i cannot be 0 for all i. As a consequence under any double error, the syndrome is different than zero and the error is detected. However, a syndrome different than zero will activate an error correction. But this action will give an incorrect result, as the code is not able to correct double errors. By adding a bit that computes the parity of the data and check bits, it is possible to distinguish single from double errors affecting the data and/or the check bits. This is because the parity bit will detect the single errors but not the double ones. The Hamming code extended with a parity bit is known as modified Hamming code.

8.2.1.2 Other SEC-DED Codes

The Hamming code has a unique characteristic: its syndrome indicates in binary code the position of the erroneous bit. However, its capability to correct single errors is not unique. Indeed, any parity code using an H matrix (the parity matrix Ps computing its syndrome) with distinct columns corrects single errors. This is because the bit i of the syndrome S_q corresponding to an error affecting the bit position q is $S_{i,q} = a_{i,q}$, where $a_{i,q}$ is the ith row element of column q of the H matrix. That is, the syndrome S_q corresponding to an error affecting the bit position q is equal to the column q of the H matrix. Thus, a parity matrix with distinct columns produces a unique syndrome for each single error.

By carefully designing the H matrix, we can optimize the code with respect to area, power, and speed. Such codes were proposed by Hsiao [13]. The parity matrix Pc generating the check bits for a Hsiao code has the following properties:

1. The columns of the matrix are distinct. This guarantees the single-error correction capability of the code.
2. Every column contains an odd number of 1s. This guarantees that the code has also double-error detecting capabilities. Indeed, this property guarantees that the syndrome of any single error has an odd number of 1s (the syndromes for single errors are done by the columns of the matrix). The syndrome for a double error is equal to the bit-wise XOR of the syndromes of the single errors composing it. Thus, it will contain an even number of 1s, allowing the distinction of double from single errors.
3. The total number of 1s in the matrix reaches is minimized. This property minimizes the number of XOR gates in the encoder and the decoder, minimizing area and power.
4. The difference of the number of 1s in any two rows is not greater than 1. The number of 1s in each row of the parity matrix Pc determines the number of inputs of the XOR tree generating the check bits. Thus, the delay of the encoder and the decoder is determined by the row comprising the maximum number of 1s.

Therefore, property (4) minimizes this maximum number as it evenly distributes the number of 1s over the rows. The work by Hsiao [13] also addresses the problem of triple-error miscorrection and quadruple-error detection, showing that the new codes behave better that the modified Hamming code. Reducing the miscorrection probability of triple errors is specifically addressed in [14]. The proposed codes minimize the triple-error miscorrection probability for the same number of check bits as the Hamming and Hsiao codes. They also feature adjacent double bit error correction capabilities.

While numerous ECC with multiple-error correction capabilities exist, the incurred area, speed and power penalties for implementing them, especially in embedded RAMS, can be quite high. As a matter of fact, codes enabling single-error correction, like Hamming or Hsiao codes, are the most used in practice for protecting memories. Such codes will be sufficient in most practical situations, since the probability that a single-event flips several cells is low. However, as we move to higher device densities, the probability of this situation increases, since memory cells are closer to each other and can be flipped by a single ionizing particle. In addition, the interaction of a neutron with silicon, oxygen, or another atom usually creates several secondary particles which may also induce multi-cell upsets. Thus, in designs targeting quite high levels of reliability, protection against multi-cell upsets has to be considered carefully. In such designs, it is still possible to use single-error correcting codes, but we must select memories using a large column multiplexing. Thus, by interleaving cells of several words in the RAM row, we increase the distance between cells of the same word and prevent multi-cell upsets affecting more than one bit in the same memory word.

8.2.1.3 Error Detection Only

The cheapest solution for protecting a memory is to add a parity bit to each memory word. During each write operation, a parity generator computes the parity bit of the data to be written. The data together with the computed parity bit are written in the memory. If a particle strike alters the state of one bit of a memory word, the error is discovered by checking the parity code during each read operation. Because this scheme detects but does not fix the error, it must be combined with a system-level approach for error recovery. This reduces the interest of the scheme, since it increases the complexity of the system design. However, in some situations, error recovery can be very simple. For instance, if the memory is an instruction cache or a write-through data cache, all the data in the cache can also be found in the main memory. Thus, the erroneous data can be recovered from this memory by simply activating the miss signal of the cache each time the parity checker detects an error. But, in situations where error recovery is more complex, it may be preferable to protect the memory by means of codes enabling error correction.

Coming back to the single-error correcting codes, for protecting a memory storing words of k data bits, these codes add r check bits, where r is the smallest

integer verifying the relation $r \geq \log_2(k + r + 1)$. It is obvious that these codes will incur high area and power penalties for memories with short words, and moderate or low area and power penalties for memories with larger words. For instance, for 4-bit word length, the area and power penalty is roughly 75%. This goes down to 50% for 8-bit word length, 31% for 16-bit word length, 19% for 32-bit word length, and 11% for 64-bit word length. This is without counting the parity bit for distinguishing single from double errors.

From the above discussion, we observe that single-error correcting codes are very convenient for modern designs using large data widths, since their cost for large word widths becomes moderate. However, some other shortcomings may make their use problematic. The first one is the speed penalty introduced by the circuitry performing check-bit computation during a write operation and error correction during a read operation. This penalty increases as the word width increases and can be a real problem, especially to those memories where ECC is most suitable, since it introduces low area and power penalty. Solutions allowing coping with this problem are therefore mandatory.

8.2.2 Removing Speed Penalty in Memories Protected by ECC

ECC introduces extra delays:

- In the path of write data due to the circuitry used to compute the check bits
- In the path of read data due to the circuitry used to compute the check bits and correct the error in the read data

It may be suitable in a design to remove the extra delay from one or both of these paths.

Figure 8.2 depicts the block diagram of an implementation eliminating extra delay in the path of the write data [15]. As shown in this figure, the data bits and the check bits are stored in two separate memories (data-memory and code-memory). We can see in the figure the block generating the check bits from the data bits to be written in the data-memory, as well as the block generating the check bits from the

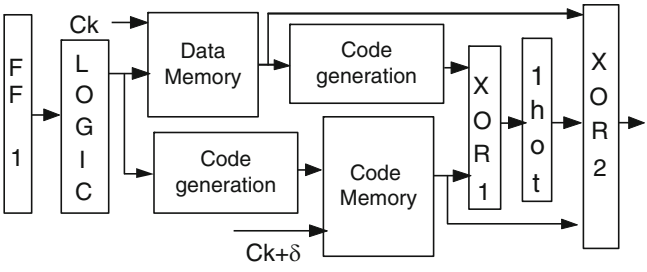


Fig. 8.2 Elimination of extra delay in the path of write data

data bits read from the data-memory. The later check bits are bit-wise XORed with the check bits read from the code-memory by a stage of XOR gates (XOR1), which produces the error syndrome. This syndrome indicates the position of the erroneous bit in a binary-code form. A combinational block transforms this form into a 1-hot code (1-hot block). Finally, a second block of XOR gates (XOR2) bit-wise XORs the 1-hot form of the syndrome with the data bits read from the data-memory and the check bits read from the code-memory to correct eventual errors affecting these bits. In this scheme, the data bits are written in the data-memory as soon as they are ready. Thus, the frequency of the clock signal Ck is the same as that in the case of a system which does not implement ECC. This way, the speed penalty on the write operations is eliminated. However, an extra delay is added on the inputs of the code-memory by the code generation block. To manage this delay, the code-memory uses a clock signal Ck + δ, which has the same frequency as the signal Ck, but it is delayed with respect to this signal by a delay δ equal to the delay of the code generation block. As a matter of fact, the check bits are ready when the signal Ck + δ enables writing these check bits in the code-memory. But, since the signal Ck + δ enables both the write and the read operations from the code-memory, then, during a read operation, the check bits will be read with a delay δ with respect to the data bits read from the data-memory. However, this delay does not increase the delay of the error correction process, because the check bits read from the code-memory are applied directly to the XOR block, while the data bits read from the data-memory have to traverse the code generation block, which has a delay equal to δ (same as the code generation block used in the write side).

The delayed clock signal Ck + δ can be generated locally, by adding a delay element on the Ck signal of the data-memory. This local generation minimizes skews between the clock signals of the data-memory and the code-memory. Careful implementation of the delay element will consider worst-case delay of the code-generation block as well as best-case (minimum) delay of this element. Another possibility is to use a single clock signal for both memories, but use one edge of this signal (e.g., the rising edge) as the active edge for the data-memory, and its second edge (e.g., the falling edge) as the active edge for the code-memory. This will work if the delay of the code generation block does not exceed the time interval that separates these edges. From the synchronization point of view, this approach is similar to the time redundancy approach using duplicated flip-flops with shifted clocks (see Sect. 8.2.3, Fig. 8.13). For the same reasons as the ones discussed in Sect. 8.2.3, the code generation block may by padded to guarantee that no path between FF1 and Code-memory has delay lower than δ.

Another implementation consists of adding a pipe-line stage in the code generation block. With this solution, the operations in the check-memory will be performed one clock cycle later than in the data-memory. This solution will work if the delay of the code generation block does not exceed one clock cycle. If this delay is larger than the clock cycle, more pipe-line stages are added. This means that a write in the code-memory will be performed more than one clock cycle after a write in the data-memory. This large delay is balanced by the delay of the code generation block placed on the read data, since, in this block, a similar number of pipe-line

stages are added. Long delays in the later block will be handled by the scheme that removes speed penalty on the read data, as described below.

Figure 8.3 depicts the block diagram of a system where the data are read from a memory pass through a combinational logic block (Logic1) and are stored in a stage of latches/flip-flops (FF1). The block Logic1 can eventually be empty. This corresponds to the case where the data read from the memory are stored directly in the FF1 stage of flip-flops. In this figure, the memory is not protected by an ECC. If ECC is used, the error detection and correction circuitry will be placed between the memory and the combinational block Logic1, increasing significantly the signal delay and decreasing accordingly the clock frequency.

In many designs, it may be required to maintain the clock frequency of the initial (unprotected) design. For doing so, one solution consists of adding a pipe-line stage (i.e., a stage of flip-flops) between the memory and the flip-flops-stage FF1 in Fig. 8.3. This solution could reach the desirable clock frequency if the delay of the detection and correction logic does not exceed the clock cycle. But the system performance will still be impacted, as the data read from the memory will reach the flip-flops-stage FF1 one clock cycle later than in the unprotected design. This may not be desirable in many designs.

A second solution is based on the observation that generating an error detection signal is faster than correcting the error. Based on this observation, the clock frequency can be determined to account only for the delay of the error detection signal. The data read from the memory are supplied to the system through a MUX. The first inputs of the MUX come directly from the memory and the second inputs from the error correction block. In error-free operation, the first inputs of the MUX are supplied to the system. When the error detection signal detects an error, the system is halted for one clock cycle to provide some extra time for performing the error correction. In the next cycle, the second inputs of the MUX are supplied to the system. While this solution reduces the speed penalty, the designer may yet not achieve its target speed due to the delay added by the error detection signal.

In processor-based systems, error recovery can be implemented at architectural level. In this case, the ECC can be checked in parallel with the operation of the pipeline, avoiding speed penalty related to error detection and correction. Then, in the absence of errors, the ECC implementation will not affect the system speed. Thus, the system operation will be delayed only when an error is detected, and the recovery mechanism is activated to resume operation after error correction.

Another scheme [15] implemented at circuit level is able to eliminate all the delays from the read port of the memory (error detection, code generation, and error

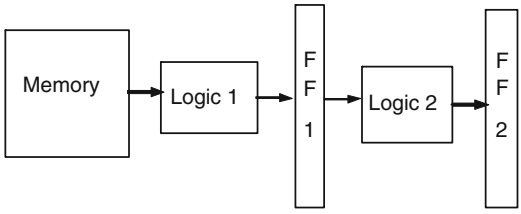


Fig. 8.3 A system with an unprotected memory

correction delays). This scheme is applicable in any kind of circuits, that is, even if the logic interacting with the memory is not a processor. It is illustrated in Fig. 8.4. This figure corresponds to the design of Fig. 8.3, but considers that the memory is protected by an ECC. For easier illustration, the detection and correction blocks are shown to be distinct, but in reality they share the code generation block and the XOR1 block. The idea here is to supply, through a MUX, the data to the system as soon as they are read from the memory. Thus, we can use a clock period duration that does not account for the error detection and error correction delays. In this case, the system will work properly as far as no errors are present in the data read from the memory. During this time, the MUX supplies to the system the data coming directly from the memory. But, when an error is present in the read-data, it is propagated and contaminates the system before the error detection signal indicates its presence. In this case, when the error detection signal is activated, it activates during the next clock cycle the hold signal of all the latches of the system but Latch1. It also enables the MUX to supply to the system the data coming from the error correction block. Thus, during one clock cycle all the latches but Latch1 are hold, maintaining their previous state, while Latch1 is decontaminated during the same cycle, since it receives the corrected data. The system restarts from a correct state on the next clock cycle. In the above, we consider that the erroneous data propagate to a single stage of latches (Latch1). But if the clock is very fast and/or the data width is large (resulting in large delay for the detection block), and/or the system is large (long interconnections for distributing the hold signal), and/or the code is complex, it may take more than one clock cycle for detecting the error and holding the system latches. In such cases, the erroneous data will be propagated in more than one pipe-line stages before the system latches are hold and the decontamination process is activated. To handle this case, the detection and correction blocks are pipelined to make them compatible with the clock cycle. The number of pipe-line stages used in the detection block also accounts for the delay of the interconnections that distribute the hold signal along the circuit. Then, as described in [15], the decontamination process will take several clock cycles (as many as the number of contaminated pipe-line stages). In addition, the number of cycles during which the latches are hold is variable. For a given latch, this number is equal to the number of pipe-line stages that separate this latch-stage from the memory.

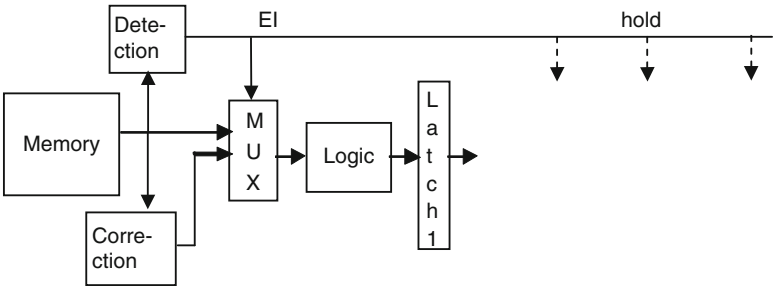


Fig. 8.4 Elimination of error detection and correction delays

The case where the logic block placed between the memory and Latch1 receives inputs from other circuits in addition to the read data is also treated in [15].

The implementations shown in Fig. 8.2 for eliminating speed penalty on the write side of the memory and in Fig. 8.4 for eliminating speed penalty from the read side were experimented over a telecoms design performing orthogonal frequency division multiplexing (OFDM).[1] Table 8.1 presents the results. Column 1 presents the different codes that were evaluated: Hamming (22, 16), Hsiao (22, 16), BCH (31, 16, 3), RS (56, 16, 17), and RS (20, 8, 17), where the third parameter in BCH (31, 16, 3), RS (56, 16, 17), and RS (20, 8, 17) gives the number of correctable bits, not the number of correctable symbols. Column 2 in the table presents the maximum operating frequency obtained with the standard implementation of these codes, and column 3 the maximum operating frequency obtained with the new implementation. Columns 4 and 5 present the extra area and extra power required by the new implementation in comparison with the standard implementation. In all cases, the frequency allowed by the new implementation is equal to 278 MHz, which is the frequency of the unprotected OFDM circuit (no ECC for memories). We observe drastic speed improvements with respect to the standard code implementations (40% for Hamming, 42% for HSIAO, and up to 329% for Reed–Solomon (56, 16, 17), while the area and power penalties with respect to the same implementations are low. Note also that for the Hamming and HSIAO codes, the erroneous data contaminate one flip-flop level before error detection and activation of the hold signal. Thus, we used one clock cycle for performing decontamination. For BCH (31, 16, 3) and RS (20, 8, 4), the erroneous data contaminate two flip-flop levels before the activation of the hold signal. Thus, we used two clock cycles for performing decontamination. Finally, for RS (56, 16, 17), the erroneous data contaminate three flip-flop levels before the activation of the hold signal. Thus, we used three clock cycles for performing decontamination. In addition, for this code, error correction delay requires inserting one wait cycle before starting decontamination. This means that at each error detection, computations will take one, two, or four extra clock cycles. However, this represents insignificant performance penalty, as error occurrences are rare (MTBF of several weeks, months, or years according to the system complexity). Thus, performance penalty is insignificant

Table 8.1 Elimination of error detection and correction delays in an OFDM circuit

OFDM design	Fr standard (MHz)	Fr new (MHz)	Area new (%)	Power new (%)
Hamming (22, 16)	198	278	+2.64	+6.01
Hsiao (22, 16)	196	278	+2.69	+3.62
BCH (31, 16, 3)	151	278	+2.15	+9.04
RS (56, 16, 17)	64.7	278	+2.24	+8.27
RS (20, 8, 4)	171	278	+1.91	+6.75

[1]The implementation of the circuit and the evaluation of the technique are done by Thierry Bonnoit, Ph.D. student at TIMA lab.

even for complex codes like BCH [102, 103] or Reed–Solomon [20]. Such codes gain interest as multiple-cell upsets (MCUs) become increasingly frequent and affect increasingly distant memory cells, making increasingly difficult to handle them by column interleaving. Furthermore in recent memory designs (e.g., 8-bit SRAM cells) used to cope with process variability, interleaving is not possible.

8.2.3 ECC Versus Non-Standard Memories

The above discussions consider memories performing read and write operation in the conventional manner. Implementing ECC in some memories performing read and write operations in non-conventional manner, such as memories with maskable operations and content addressable memories (CAMs), can be more difficult.

In a memory allowing maskable operations, some write operations will write data on all bits of the word selected by the current memory address, but some other writes may write data only on a subset of bits of the memory word. This subset of bits is determined by setting the bits of a mask. When a subset of bits of a word is written, the contents of the remaining bits are unknown. Thus, it is not possible to compute the new check bits for this word [16]. To cope with this problem, several solutions are possible [16].

The first one consists of coding separately several subsets of the bits of each word. This is possible because usually the maskable writes are not performed over any random collection of bits of a word. For instance, for a memory using 64-bit words, the word can be divided into eight fields of 8 bits and the maskable writes are performed over one of these fields. In this example, instead of implementing a single code for the whole word, we can implement one code for each one of the above eight fields. Because during a maskable or a non-maskable write, all the bits of each written field are known, the check bits for each written field can be computed. While this solution enables implementing the code, the area and power cost can be significant because the size of the encoded data is small. For instance, in the above example, we will need to add 50% extra bits to store the check bits in the memory (4 check bits for every 8 data bits). The cost is even higher if we use a code able to differentiate between single errors and double errors (5 check bits). The worst case corresponds to memories where each maskable write is performed over a single bit. In this case, the above solution leads on triplication of the memory bits.

Another solution, which allows eliminating this high cost, consists of performing a read before each maskable write, correct the error in the read data, if any, and combine the read and the written data to compute the new check bits. This solution reduces the cost for protecting memories with maskable operations to the cost required for regular memories. However, performing an extra read implies a performance penalty which may not be acceptable in many systems.

A third solution consists of exploiting the following fact: when a maskable write is performed, the bit-lines of the bits which are not affected by this write are precharged and the amplifiers of these bits are in the read mode. In this situation,

a simple modification of the memory allows reading these bits simultaneously with the write operation performed over the bits selected by the maskable write. In this manner, we dispose the values of all the bits of the memory word (i.e., the values written in the bits selected by the maskable write and the values of the non-selected bits). Thus, we can compute the check bits for the actual data stored in this word and write these check bits in a separate memory (code-memory). Because the write of certain bits of the word and the read of certain other bits are performed in parallel, there is no performance penalty. However, we need to cope with a major difficulty: the read bits may include errors. These errors must be corrected before computing the new check bits, otherwise the check bits computation will mask the errors. For detecting and correcting these errors, we need to know the values stored in every bit of the word before performing the maskable write. This means that we need to read all the bits of the word, but as described above, we only read the bits not selected by the maskable write. So, detecting and correcting an error on these bits are not possible by using conventional coding schemes. A non-conventional solution is proposed in [16]. The simplest implementation corresponds to memories which perform maskable writes over fields composed of a single bit. In this case, using a single-error correcting/double-error detecting code (e.g., the Hamming code and a parity bit) and a special error correction circuitry allows correcting any single error in the read bits. So, in this case, the cost of the check bits is the same as that for the conventional codes. When the maskable operations use larger bit fields, a more complex code is required. This code adds $p + 2$ extra check bits, where p is the size of the field on which a maskable operation is performed. The details can be found in [16].

This approach can also be used in memories where maskable writes can be performed over random collections of bits, in which case the scheme implementing one code per bit field is not applicable. For such memories, the use of the approach performing parallel reads and writes of the bits of a word requires implementing the memory in a manner that several words can be read in parallel. This can be done by selecting an appropriate memory size. For instance, for a memory with 16-bit words, we can use a memory with 64-bit words but having a number of words four times less than the initial memory. Operations in the new memory are performed in a tricky manner that makes them equivalent to the operations performed in the initial 16-bits memory, and at the same time, it allows proper code computation for maskable and non-maskable operations. The scheme allows a drastic reduction of the check bits. For instance, for a memory using 16-bit words and performing maskable writes over random bits, the only alternative solution to the new codes is triplication (i.e., 200% of extra bits). If we implement the new code by using a memory that allows reading in parallel 64 bits (four 16-bit words), the extra bits represent 39% of the bits of the unprotected memory. But this number has to be considered carefully, since the area overhead will be higher than 39%. In fact, with this scheme, in order to perform the code computation process properly, the check bits have to be stored in a separate dual-port memory allowing simultaneous read and write of two words. The higher area occupied by such memories will lead for this example to a total extra area of 60–70%. But still, this

area remains much lower than the 200% extra area required if we use the only other alternative solution.

Content addressable memories are another kind of memories that are difficult to protect. Each word in such memories comprises a data field and an address field (or tag field). The address field of each word includes a comparator which allows comparing in parallel all the addresses stored in the CAM against an address applied on the address inputs of the CAM (associative search). When the address stored in one of the CAM words matches the externally applied address, a hit is activated and the data stored in the data field of this word are read or new data are written. While many varieties of CAMs exist, including ternary CAMs or TCAMS that allow masking a selected set of bits of the address field of a word or of the externally applied address, all varieties share a common characteristic that makes difficult the complete protection of a CAM. Indeed, when an error affects the address stored in a CAM word, an associative search of the correct address value will produce an incorrect miss. This error cannot be detected, since, as a consequence of the incorrect miss, the word containing the erroneous address is not read to check for its integrity. As a matter of fact, we can use parity or ECC to protect the data field and the address field of a CAM. However, while the data field will be completely protected, the address field will be protected only when the error leads to an incorrect hit, since in this case, the hit word can be read and checked, but it will not be protected when the error leads to an incorrect miss. An incorrect miss will not be a problem in systems where the CAM is used as a cache. In this case, a miss enables retrieving the data from the main memory. Unfortunately, there are numerous systems employing CAMs where this is not the case.

The above discussions illustrate that for some memory types, it may be quite expensive or even impossible to achieve protection against soft errors by means of ECC. In addition, ECC may introduce a non-negligible area cost and significant speed penalty even for standard SRAM memories. Also, the reach of multiple-cell upsets (MCUs) is widening and their FIT is steadily increasing. Thus, it becomes increasingly difficult to avoid multiple-bit upsets (MBUs) by using interleaving. Also, interleaving is not possible in some recent SRAMs using specific 8-transistor cells for mitigating process variability. In this context, alternative approaches could be used, including multiple-error correcting codes like Reed–Solomon. As these codes require a large number of check bits, alternative solutions can be used like Built-In Current Sensors (BICSs) to detect and correct single and multiple errors in memories [17–19], and hardened cells to avoid SEUs in memories and in sequential logic cells such as latches and flip-flops. These techniques are presented next.

8.3 CRC Codes

Cyclic redundancy check (CRC) codes are best suited and commonly used for protecting data packets in transmission channels (e.g., CRC-32-IEEE 802.3 standard) and mass storage devices (e.g., encoding of disc sectors by means of the

CRC-CCITT). CRC codes have mathematical properties, allowing easy and efficient design of codes with desirable error detection capabilities. Another interest is that they require very simple circuit for encoding and decoding. Beyond error detection, any standard CRC code could theoretically be used for correcting single errors (any two different single errors have different syndrome). But in practice, CRC codes are commonly used for error detection purposes, while error correction is done by different means. For instance, to perform error correction in transmission channels, after CRC-based error detection, the receiver can send a retransmission request to the transmitter, whereas in systems where the lack of acknowledgment leads to a retransmission, the receiver just omits sending a receipt acknowledgment (a technique known as Automatic Repeat reQuest – ARQ).

On the contrary, in mass storage devices, error detection is performed by some other ECC (e.g., other parity codes, Reed–Solomon, etc.). In addition, CRC can also be used either for signaling uncorrectable errors that could result in miscorrections, or for indicating the location of the erroneous symbols (erasures) and increasing the error correcting capability of the ECC (e.g., Reed–Solomon codes correct twice as many erasures as errors affecting unknown locations).

CRC in mass storage devices and transmission channels is not used for protection against radiation-induced soft errors, as these media are insensitive to this issue. But as a by-product of the protection of transmission channel/mass storage media, CRC can also be used to protect temporal storage hardware (e.g., FIFO buffers) on the emission/write port and reception/read port of transmission channels/storage media at almost no cost. This kind of hardware is of course sensitive to soft errors.

CRC and Reed–Solomon codes are linear codes and can be described by means of parity matrixes. In the following, we will describe their underlying theory, based on Galois field GF(2^m) algebra without presenting their representation with parity matrixes. Interested readers can find more details on these codes in the literature [20–22].

A CRC code is computed by representing k-bit binary words as polynomials of degree $k - 1$ over GF(2) (i.e., polynomials with coefficients equal to 0 or 1). More precisely, the coefficients of the polynomial are equal to the bits of the binary word. That is, the polynomial corresponding to the binary word $D_{k-1}2^{k-1}, D_{k-2}2^{k-2}, \ldots, D_1 2^1, D_0 2^0$, is $D_{k-1}X^{k-1} + D_{k-2}X^{k-2} + \cdots + D_1X^1 + D_0X^0$. For instance, the polynomial corresponding to the binary word 10100101 is $X^7 + X^5 + X^2 + X^0$. The CRC code is determined by its generator polynomial $G(x)$. To form the code for a data word, we extend this word by a check part determined as the reminder $R(x)$ of the division of $D(X)X^r$ by $G(x)$, where $D(X)$ is the polynomial associated to the data word and r is the degree of $G(X)$.

As an example, let us consider the binary word 10100101. Then, the corresponding polynomial is $D(X) = X^7 + X^5 + X^2 + X^0$. Let $G(X) = X^3 + X^2 + X^0$ be the generator polynomial (1101 in binary). Then, $D(X)X^r$ is written in binary as 10100101000. To perform the division of $D(X)X^r$ by $G(x)$, we need only the coefficients of the polynomials. Thus, we can work on their binary representations. In GF(2), the sum of two binary values corresponds to their XOR. Then, the

division of 10100101000 by 1101 shown below gives $R(X) = 0001$, which is the check part of the binary word 10100101.

```
10100101000 | 1101
1101          11010101 → Q(X)
01110
 1101
 001110
  1101
  001110
   1101
   001100
    1101
    0001 → R(X)
```

The message transmitted comprises $k + r$ bits and consists of the concatenation of the data bits with the check bits. This corresponds to the polynomial $T(X) = D(X)X^r + R(X)$. But, since in modulo 2 sum, addition and subtraction are equivalent, we find $T(X) = D(X)X^r - R(X) = G(X)Q(X)$. On the reception side, the received message is checked by computing the reminder of the division of $T(X)$ by $G(X)$. If there are no errors in the transmitted message, this reminder will be zero as $T(X)$ was found to be equal to $G(X)Q(X)$.

One of the major interests of the CRC codes is the easy encoding and decoding, by means of a linear feedback shift register (LFSR). This encoding is of serial nature, thus it fits well with serial data transmission. An LFSR comprising r flip-flops can be associated to any polynomial of degree r that has non-0 coefficient for the term X^0 (i.e., a polynomial of the form $G(X) = X^r + g_{r-1}X^{r-1} + g_{r-2}X^{r-2} + \cdots + g_1X^1 + X^0$). Figure 8.5 presents the LFSR corresponding to this polynomial. In this figure, $g_i = 0$ means that there is no feedback from the output (signal Out) to the ith stage (the corresponding XOR gate and its input from Out are omitted), while $g_i = 1$ means that there is a feedback from Out to the ith stage (the corresponding XOR gate and its input coming from Out are effectively realized). Each of the coefficients g_1 through to g_{r-1} is either 0 or 1. It can be shown that shifting $k + r$ times a k-bit data word in the LFSR from its external input "In" computes the division $D(X)X^r/G(X)$, with the quotient of the division appearing in the output "Out" during the last k shifts, while the LFSR at the end of the $k + r$ shifts contains the reminder $R(X)$. Thus, if we use an LFSR corresponding to the generator polynomial of a CRC code, the content of the LFSR at the end of the $k + r$ shifts will provide the check part of the code. The LFSR shown in Fig. 8.5 is referred as internal XOR LFSR. There is also another type

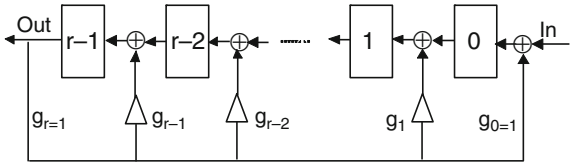


Fig. 8.5 Internal XOR LFSR for characteristic polynomial $X^r + g_{r-1}X^{r-1} + g_{r-2}X^{r-2} + \cdots + g_1X^1 + X^0$

of LFSR referred as external XOR LFSR. This LFSR type is slower as its XOR gates are connected serially and involve larger delays.

It is easy to check that polynomial division defined above is linear. Thus, the reminder of a polynomial $T(X) \oplus E(X)$ is equal to $\mathrm{Rt}(X) \oplus \mathrm{Re}(X)$, where $\mathrm{Rt}(X)$ is the reminder of the division of $T(X)$ by $G(X)$ and $\mathrm{Re}(X)$ is the reminder of the division of $E(X)$ by $G(X)$. Thus, if the transmitted message is affected by an error represented by the polynomial $E(X)$, the error is detected if and only if $G(X)$ does not divide $E(X)$. Thus, the error detection capabilities of the CRC code depend on the properties of the generator polynomial used. An interesting class of polynomials consists of primitive ones. A primitive polynomial is a prime polynomial of degree r which divides the polynomial $X^{\mathrm{T}} + 1$ (with $T = 2^r - 1$) and does not divide any other polynomial $X^s + 1, 0 \leq s < T$. Primitive polynomials over GF(2) of degree up to 34 can be found in [23]. It follows directly from this property that a linear cyclic code whose generator polynomial $G(X)$ is primitive detects all single errors and all double errors within a message of length $2^r - 1$. Indeed, a primitive polynomial $G(X)$ of degree r has always the form $X^r + g_{r-1}X^{r-1} + g_{r-2}X^{r-2} + \cdots + g_1X^1 + X^0$ (the coefficient of X^r is not 0 since the degree of $G(X)$ is r, and the coefficient of X^0 is not 0 since in this case $G(X)$ will be divided by X, which is not possible since $G(X)$ is prime). From this form it follows that all multiples of $G(X)$ have at least two terms. Thus, $G(X)$ cannot divide an error polynomial $E(X) = X^j$, meaning that the corresponding CRC detects all single errors in a transmitted message of any length.

The polynomial of a double error within a string of length $2^r - 1$ can be written as $E(X) = X^i + X^j$, with $j < i$, $i < 2^r - 1$. Then, $E(X) = (X^{i-j} + 1)X^j$. We have $i - j < 2^r - 1$, and from the definition of primitive polynomials, $X^{i-j} + 1$ is not divided by $E(X)$. We have also found that X^j is not divided by $G(X)$. Hence, $E(X)$ is not divided by $G(X)$ and the code detects any double error within any message of length $2^r - 1$ bits or less. This means that in order to detect all single and double errors in a message of k information bits, we can select as generator polynomial a primitive polynomial of degree r, with $r \geq \log_2(k + 1)$.

Concerning errors affecting t consecutive bits (i.e., burst errors of length t), their error polynomial can be written as $B(X)X^j$, where the degree of $B(X)$ is equal to $t - 1$. As the reminder of the division of $B(X)X^j$ by $G(X)$, for any value of j, cannot be 0 if the degree of $B(X)$ is lower than r (since $G(X)$ will not divide $B(X)$ or X^j), the code detects all burst errors of length lower than or equal to r.

Another interesting property is that any multiple of $X + 1$ ($X + 1$ is often referred as *parity polynomial*) has an even number of terms. Thus, by selecting as generator polynomial $G(X) = (X + 1)H(X)$, with $H(X)$ a primitive polynomial of degree r, we can detect all single errors, all double errors within a message of length $2^r - 1$ bits or less, all burst errors of length r or less, and all errors of odd multiplicity. These error detection capabilities are common in any well-constructed CRC generator polynomial. But other types of errors may go undetected. The probability for a random error affecting a message of n bits to go undetected by a generator polynomial of length r is $(2^{n-r} - 1)/(2^n - 1) \approx 2^{-r}$. For a generator polynomial of degree 16, this probability is $\approx$1/65,536 $\approx 1.5 \times 10^{-5}$. The number

of errors that can go undetected by a generator polynomial is referred to as Hamming weight. Efforts for improving error detection probability of CRC codes consist of determining generator polynomials with low (possibly zero) Hamming weight for errors of certain multiplicity affecting a message of a given length. Indeed errors of even multiplicity higher than 3 are of interest, as single and double errors are detected by selecting a primitive generator polynomial and all odd multiplicity errors are detected by multiplying it with the parity polynomial $X + 1$. A comprehensive study concerning the error detection capabilities of CRC codes and the selection of the most efficient generator polynomials can be found in [24].

8.4 Reed–Solomon Codes

Reed–Solomon codes [20] compose a class of powerful codes correcting errors over symbols taking 2^m values rather than errors over single bits. A Reed–Solomon code is defined by three parameters n, k, and t, and any positive integer $m > 2$ such that $(n, k) = (2^m - 1, 2^m - 1 - 2t)$, where the total length of the code word is $n = 2^m - 1$ symbols, k is the number of data symbols, $n - k = 2t$ is the number of check symbols, and the symbol-error correcting capability of the code is equal to t. For instance, a (7, 3) Reed–Solomon code has three data symbols and four check symbols, and can correct up to two erroneous symbols $(t = (7 - 3)/2 = 2)$.

The definition of Reed–Solomon codes uses Galois field GF(2^m) algebra. GF(2^m) has 2^m elements $\{0, \alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{2^m-2}\}$. The element α^i is associated to the polynomial X^i modulo $f(X)$, where $f(X)$ is a primitive polynomial of degree m defined in the Galois field GF(2).

Since by definition a primitive polynomial of degree m does not divide any polynomial $X^s + 1$, for $0 \leq s < 2^m - 1$, $f(X)$ will not divide any polynomial $X^p + X^q$ for $0 \leq p < 2^m - 1$, $0 \leq q < 2^m - 1$ (as we have seen earlier). Therefore, $X^p \text{ modulo } f(X) \neq X^q \text{ modulo } f(X)$, for $p \neq q$, $0 \leq p < 2^m - 1$, $0 \leq q < 2^m - 1$. From this property, we find easily the following properties for the elements of GF(2^m):

1. The elements $\alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{2^m-2}$, produced by the operation X^i modulo $f(X)$, are distinct.
2. Each of these elements corresponds to one of the $2^m - 1$ distinct polynomials of degree lower than m.
3. The first m elements are the monomials $\alpha^0 = X^0 = 1, \alpha^1 = X^1, \alpha^2 = X^2, \ldots, \alpha^{m-1} = X^{m-1}$.
4. Since the remaining elements $\alpha^m, \alpha^{m+1}, \ldots, \alpha^{2^m-2}$ are polynomials of degree lower than m, they can be expressed as sums of the first m ones.
5. The addition of any two elements $\alpha^p + \alpha^q$ is obtained by adding the corresponding polynomials. The result can always be reduced into a single element α^i.

This is because the addition of two polynomials of degree lower than m is a polynomial of degree lower than m. Since to each polynomial of degree lower than m corresponds an element α^i (property 2), $\alpha^p + \alpha^q$ can always be reduced into a single element α^i. This property determines the operation of addition of the elements of GF(2^m).

6. The multiplication $\alpha^p\alpha^q$ of any two elements α^p and α^q is obtained by multiplying the corresponding polynomials and taking the modulo $f(X)$ of the result. We find that $\alpha^p\alpha^q$ is equal to α^{p+q}. The powers α^q with q higher than $2^m - 2$ are defined as $\alpha^q = \alpha^{q \text{ modulo } 2^m-1}$. That is, $\alpha^{2^m-1} = \alpha^0$, $\alpha^{2^m} = \alpha^1$, etc. This property allows completely defining the multiplication between the elements of GF(2^m).

Example. Let us consider the Galois field GF(2^3) corresponding to the primitive polynomial $X^3 + X + 1$:

- From the previous discussion, we will have $\alpha^0 = X^0 = 1$; $\alpha = X$; $\alpha^2 = X^2$; $\alpha^3 = X^3 \text{ modulo}(X^3 + X + 1) = 1 + X = 1 + \alpha$; $\alpha^4 = X^4 \text{ modulo}(X^3 + X + 1) = X + X^2 = \alpha + \alpha^2$; $\alpha^5 = X^5 \text{ modulo}(X^3 + X + 1) = 1 + X + X^2 = 1 + \alpha + \alpha^2$; $\alpha^6 = X^6 \text{ modulo}(X^3 + X + 1) = 1 + X^2 = 1 + \alpha^2$. We can rewrite these relationships together with the 3-tuple corresponding to each polynomial. Thus, we obtain the relationships: $\alpha^0 = X^0 = 1$; $\alpha = X$; $\alpha^2 = X^2$; $\alpha^3 = 1 + X = 1 + \alpha$; $\alpha^4 = X + X^2 = \alpha + \alpha^2$; $\alpha^5 = 1 + X + X^2 = 1 + \alpha + \alpha^2$; $\alpha^6 = 1 + X^2 = 1 + \alpha^2$, which verify the properties 1–4.
- Some illustrations for property 5: $\alpha^5 + \alpha^2 = 1 + \alpha + \alpha^2 + \alpha^2 = 1 + \alpha = \alpha^3$; $\alpha^6 + \alpha^3 = 1 + \alpha^2 + 1 + \alpha = \alpha^2 + \alpha = \alpha^4$; $\alpha + \alpha^4 = \alpha + \alpha + \alpha^2 = \alpha^2$; etc.
- Some illustrations for property 6: $\alpha^3\alpha^2 = (1 + X)X^2 = X^2 + X^3 = X^2 + 1 + X = \alpha^5 = \alpha^{3+2}$; $\alpha^5\alpha^2 = (1 + X + X^2)X^2 = X^2 + X^3 + X^4 = X^2 + 1 + X + X + X^2 = 1 = \alpha^0 = \alpha^{(5+2)\text{modulo } 7}$; $\alpha^6\alpha^3 = (1 + X^2)(1 + X) = 1 + X^2 + X + X^3 = 1 + X^2 + X + 1 + X = X^2 = \alpha^2 = \alpha^{(6+3)\text{modulo } 7}$.

A convenient way is to represent the elements $\alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{2^m-2}$ as m-bit symbols, by using the m-tuples describing their associated polynomials. For the above example, we will have $\alpha^0 = X^0 = (100)$; $\alpha = X = (010)$; $\alpha^2 = X^2 = (001)$; $\alpha^3 = 1 + X = (110)$; $\alpha^4 = X + X^2 = (011)$; $\alpha^5 = 1 + X + X^2 = (111)$; $\alpha^6 = 1 + X^2 = (101)$.

8.4.1 Encoding

Once a GF(2^m) field is determined as above (i.e., its elements as well as the addition and multiplication operations), we can define Reed–Solomon codes in this field by defining their generator polynomial $g(X)$. The degree of this polynomial is equal to the number of check symbols ($2t$) of the code and its roots are $\alpha, \alpha^2, \ldots, \alpha^{2t}$. This determines $g(X)$ as $g(X) = (X + \alpha)(X + \alpha^2)\cdots(X + \alpha^{2t})$, considering that in modulo 2 arithmetic addition and subtraction are equivalent. Then, we can obtain trivially the coefficients g_i of $g(X) = g_0 + g_1X + \cdots + g_{2t-1}X^{2t-1} + X^{2t}$.

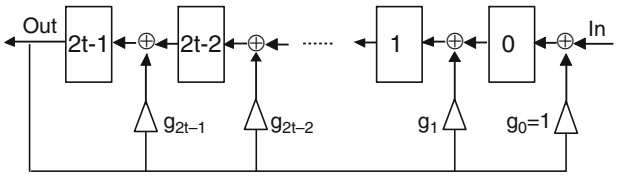


Fig. 8.6 Generalized LFSR for polynomial division in GF(2^m)

For instance, to generate a (7, 3) Reed–Solomon code using the GF(2^3) defined above, we will have $2t = 7 - 3 = 4$. Thus, $g(X) = (X + \alpha)(X + \alpha^2)(X + \alpha^3)(X + \alpha^4) = \alpha^3 + \alpha^1 X + \alpha^0 X^2 + \alpha^3 X^3 + X^4$.

The check part of the code (composed of $2t$ m-bit check symbols) for a message $D(X)$ composed of k m-bit data symbols is generated by computing the reminder of the division of $X^{2t}D(X)$ by $g(X)$. That is, $p(X) = X^{2t}D(X)$ modulo $g(X)$. This can be performed by a generalized LFSR implementing the polynomial $g(X)$ as shown in Fig. 8.6. This is similar to the generation of CRC codes discussed earlier. The difference is that the coefficients of the polynomials and of the LFSR are m-bit symbols instead of single bits, and each stage of the LFSR comprises m flip-flops instead of 1 flip-flop in the GF(2) LFSR. Also, instead of the modulo 2 arithmetic used in GF(2), the more complex arithmetic of GF(2^m) fields defined above is used.

The check part of the code is the remainder $R(X)$ of the division of $X^{2t}D(X)$ by $f(X)$ (it is computed by shifting in the LFSR the polynomial $X^{2t}D(X)$). The code word $U(X)$ is obtained by concatenating the $X^{2t}D(X)$ with $R(X)$. This code word is transmitted to the receiver (transmission channel application) or written to the memory (mass memory protection application).

8.4.2 Syndrome Computation

In the receiver/memory read side, the received/read message is $U'(X) = U(X) + E(X)$, where $E(X)$ is the error signal. From the encoding principle, we have $U(X) = X^{2t}D(X) + R(X)$ and $X^{2t}D(X) = g(X)Q(X) + R(X)$. Thus, $U(X) = g(X)Q(X)$, giving $U'(X) = g(X)Q(X) + E(X)$. From the definition of $g(X)$ as $g(X) = (X + \alpha)(X + \alpha^2) \cdots (X + \alpha^{2t})$, we have $g(\alpha^i) = 0$, $\forall i$: $0 \leq i \leq 2t$. Thus, we obtain $2t$ syndrome equations involving the error polynomial $E(X)$: $S(\alpha^i) = U'(\alpha^i) = E(\alpha^i)$, $\forall i$: $0 \leq i \leq 2t$, which defines $2t$ syndromes. $E(X)$ can be expressed as $e_{q_1}X^{q_1} + e_{q_2}X^{q_2} + \cdots + e_{q_s}X^{q_s}$, where s is the number of erroneous symbols in the received message $U'(X)$, X^{qi} determines the position of the ith erroneous symbol, and e_{q_i} is the error in this symbol. Thus, we have $2s$ unknowns (s error positions and s erroneous symbol values). If $s \leq t$, we can resolve the $2t$ syndrome equations to compute them. The techniques used for resolving these equations can be found in [21, 25].

8.5 Memory Protection Using Built-In Current Sensors

A memory cell being in the steady state drives a very small current. But when its state is reversed due to the impact of an ionized particle, abnormal activity will be observed in the cell and abnormal current will flow through the Vdd and Gnd lines of the cell. It is then possible to detect the occurrence of an SEU by implementing one current sensor (BICS) on each vertical power line of the memory cell array [17–19] as shown in Fig. 8.7. In this figure, the primary power line (Gnd or Vdd) is connected to the vertical (secondary) power lines of the memory through several transistors (NMOS if we work with Gnd lines, PMOS if we work with Vdd lines). Note that such transistors are often used to reduce static power dissipation during the idle periods of the memory (sleep transistors). Thus, they can be used for both power reduction and BICS implementation. The gates of the NMOS/PMOS transistors are set to 1/0. Each BICS monitors vertical power lines and detects SEUs affecting cells connected to these lines.

Because the BICSs monitor the vertical power lines of the memory, the detection of an SEU also indicates the position of the affected bit. This will also be true when two neighbor columns share the same power line if the memory uses some interleaving (1-out-of-k column multiplexing, $k > 2$). In such a case, neighbor cells belong to different words and an error signaled by a BICS determines which bit is possibly affected in each of these words. It is then possible to add a parity bit to each memory word, and use it for error detection each time a faulty word is read. Then the error is corrected by flipping the value of the bit indicated by the active BICS. Double errors can also be corrected if we use a single error-detecting double-error correcting (SED-DEC) code, like the Hamming or HSIAO codes. Indeed, single errors are corrected by the SED-DEC code. Furthermore, if a word containing a double error is read from the

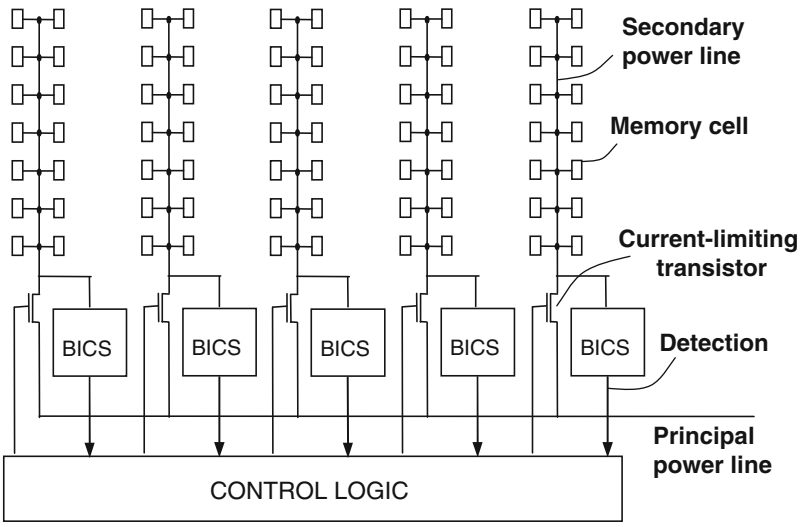


Fig. 8.7 Memory protection using Built-In Current Sensors

memory, the SED-DEC code will signal its presence. In addition, the erroneous bit positions will be known, as two BICSs will be activated at the time of occurrence of the upsets affecting these bits. Thus, these positions are flipped to correct the double error and the correctness of the resulting word can be cross-checked by the SED-DEC code.

The BICS approach can be used for standard SRAMs as well as for memories difficult to protect by ECC, such as memories using maskable operations or CAMs. In particular, in the case of CAMs, during associative searches, the address field is neither read nor written. Thus, it cannot be protected by ECC. To address this problem, we can protect the address field of a CAM by means of BICS and ECC. Each time a BICS is activated, we read the address fields of the CAM, and correct them by means of ECC. In large CAMs composed of several smaller CAM blocks, the activation of a BICS will also indicate the affected CAM block, allowing for a faster location of the erroneous word.

The BICS scheme shown in Fig. 8.7 is also efficient for detecting and eliminating single-event latchups (SEL), and correcting the induced errors [26, 27]. A latchup activates a parasitic PNPN structure (thyristor) and creates a short between the Vdd and ground lines. Thus, it will drive a large current and the BICS connected to the affected secondary power line (see Fig. 8.7) will detect the event. Then, the control logic will drive to the off state the sleep transistor associated to the active BICS and will cut the current flowing through the activated thyristor. Thus, the latchup is eliminated and the circuit can be driven back to its normal state (the control logic will drive the sleep transistor back to the on state). Furthermore, as there is a transistor between the principal power line and the secondary one affected by the latchup, the current flowing through this transistor cannot affect the voltage level of the principal power line adversely. Thus, the state of cells connected to different secondary power lines will not be affected. Therefore, only single errors will affect the words of the memory. These errors are correctable by an ECC or by the combination of the BICS's indications and a parity bit, as discussed earlier. The technique can be used for correcting errors due to both SEUs and latchups. However, in this case, it will be useful to implement a BICS able to differentiate errors due to SEUs from errors due to latchups (to initiate scrubbing when a latchup is detected). This distinction is feasible, since the current driven by a latchup is much stronger. Also, in this case, correction of double errors due to the combination of latchups and SEUs is possible by combining the BICS indications with a SED-DEC code as discussed earlier.

It is also worth noting that the principles of this scheme can also be used to eliminate latchup and correct the related errors in logic [28]. As in the case of memories, the scheme can be combined with sleep transistors employed to reduce static power.

8.6 Error Mitigation in Logic

As stated earlier in this chapter and discussed in more details in Chaps. 1 and 2, logic is affected by SEU- and SET-related soft errors. SEUs occur when an ionizing particle striking a sensitive node of a flip-flop or a latch cell flips the state of the cell.

SET-related soft errors occur when a transient pulse, initiated by an ionizing particle striking a sensitive node of a logic gate, is propagated through the gates of the combinational logic and is captured by a sequential element such as a latch or a flip-flop. The largest contributor to logic SER comes from SEUs, though SETs' contribution is gaining importance as we move to higher integration densities. As a matter of fact, if in a design the logic SER exceeds the maximum allocated FIT, protecting latches and/or flip-flops may be sufficient to achieve acceptable FIT. This could be done by replacing a set of selected latches/flip-flops by hardened ones. However, if SEU mitigation is not enough for meeting the target FIT, a more complete soft-error mitigation approach has to be used to protect logic parts from both SEUs and SETs. This approach can be based on hardware (space) redundancy and/or time redundancy.

Note also that soft errors may have a severe impact on designs implemented in SRAM-based FPGAs. In this case, the function of the design is determined by the values stored in the configuration SRAM cells of the FPGA. Thus, an SEU affecting such a cell has a much more severe impact than SEUs affecting memories, latches, and flip-flops, as it modifies the very function of the design. A solution for fixing this problem consists of reading periodically the configuration memory, checking its contents, and reprograming the FPGA when an error is detected (read-reprogram approach). This technique is acceptable only if the application tolerates long error latencies. If this is not acceptable, TMR is necessary. TMR can also be combined with the read-reprogram scheme to avoid error accumulation in the configuration memory. The interest of the TMR approach is its flexibility, as every designer can employ it. But its major drawback is a large hardware overhead, which significantly exceeds 200%. Using hardened SRAM cells to implement the configuration memory of FPGAs is the less area and power consuming solution for solving this problem. Its major drawback is a low flexibility, since it cannot be implemented on a standard FPGA family by a designer. Instead, it requires the commercialization by FPGA providers of FPGAs using hardened configuration SRAM cells.

8.6.1 Hardened Storage Cells

Hardened storage cells (SRAM cells, latches, and flip-flops) preserve their state even if the state of one of their nodes is altered by an ionizing particle strike. Various hardened storage cells have been proposed in the literature. We can consider three kinds of hardened storage cells.

The robustness of the first kind relies on adding resistors or capacitances on the feedback loop of the cell to increase its critical charge. Adding extra resistors can be realized by passive components (for instance, highly resistive polysilicon) [29]. This option allows realizing resistors of high value at low silicon area. Increasing the coupling capacitance between the storage nodes of a cell can be done without significant area impact by adding DRAM-like stacked capacitors on top of memory

cells [11]. The drawback of the above hardening approaches is that they require extra process steps that can have an impact on fabrication cost. In addition to the cost issue, such techniques may also have an impact on cell speed and power. These issues may reduce the interest of the above approaches for some commercial applications. Extra capacitances can also be added by using extra transistors and connecting their gates to the cell nodes (exploiting this way the gate capacitance of CMOS transistors), or by adding metal–metal capacitances on top of the cells. In addition to the impact on performance, the drawbacks of the later approaches are a limited reduction of the SER and the consumption of a significant area and/or the loss of two metal layers on the top of the cells.

The second kind of hardened cells uses extra active transistors in the cell, but its robustness relies on specific transistor sizing. As a result, these cells do not scale easily as the device size is shrinking. The area cost of these cells may also be high. The early hardened cells [30, 31] belong to this kind.

Figure 8.8 presents the hardened cell proposed in [30]. To modify the value of the cell, signal C (the clock signal of a latch or the world-line of a memory) is set high. This turns on transistors N3 and N4 and sets nodes X_1 and X_2 on their new values. For instance, to write a 1 in the cell, X_1 is set to 1, and X_2 to 0. This sets the cross-coupled inverters composed of transistors N1, P1, N2, and P2 to their new state, in a similar manner as in a conventional memory cell. The other nodes are set to their new state after C is turned low. In this state, the value 0 of C drives X_4 to 0 (X_2 was set to 0, thus P4 is on). This turns P7 on, and sets X_3 high. It also turns P5 on, which is compatible with the on state of P1 (both drive X_1 high). X_3 high turns P6 and P8 off, which is compatible with the low value of X_2 and X_4. Symmetrical situation holds when writing the opposite value. When C is 0, the cell should maintain its state, even if a transient pulse affects any of the nodes X_1, X_2, X_3, and X_4. Consider the case $X_1 = 1$, $X_2 = 0$, $X_3 = 1$, and $X_4 = 0$ (the other case is symmetrical and can be treated similarly). A transient pulse on X_3 can be produced by a particle striking transistor P3 or P7. As particles striking a p-type transistor can produce only positive pulses and X_3 is at 1, particle strikes cannot affect the state of

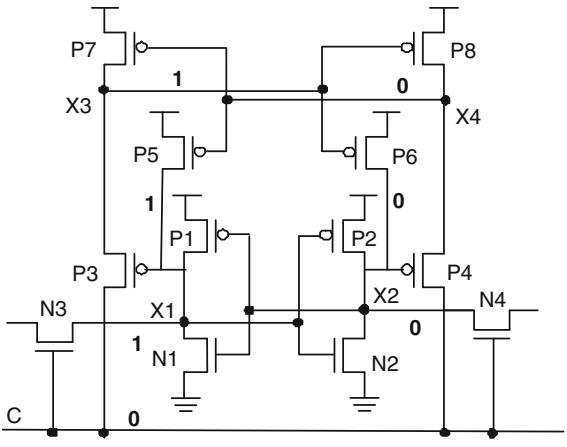


Fig. 8.8 A hardened cell [30]

X_3. A negative pulse on X_1 is problematic as it turns P2 on and N2 off, driving X_2 to 1, which turns P1 off and N1 on and reinforces the value change of X_2. Also, P3 is turned on as X_1 goes low, thus driving X_3 low. The later turns P6 on and reinforces the high level on X_2. X_3 low also turns P8 on, which drives X_4 high and turns P5 off. To improve the robustness of the cell, transistor P7 is designed much larger than P3. Thus, X_3 is maintained high, transistor P6 is maintained off and P5 on. The current flowing through the on transistor P5 drives node X_1 back to 1 and subsequently node X_2 back to 0, restoring the cell state. The cell is even more robust for the positive pulse affecting X_2. This is because, in order to turn off transistor P5, the pulse needs to be propagated one extra stage with respect to the previous case (dive X_1 low through N1, P1). A positive pulse on X_4 drives P5 and P7 off and leaves the values of nodes X_3, X_1, and X_2 unaffected. The signal C drives X_4 back to 0 through the on transistor P4.

The third kind of hardened cells does not rely on transistor sizing, thus allowing easier scaling. They also induce lower area and power penalties. The principle used in the DICE cell [32] is of this kind. It achieves immunity against transients affecting any single node without requiring any particular transistor sizing. Therefore, its scaling does not require particular care and can be implemented by using minimum transistor size and in any process generation. This makes the DICE cell more suitable for protecting storage cells against SEUs in advanced nanometric technologies. The transistor diagram of this cell is shown in Fig. 8.9. The cell has two states, the 0 state ($X_1 = 0$, $X_2 = 1$, $X_3 = 0$, and $X_4 = 1$) and the 1 state ($X_1 = 1$, $X_2 = 0$, $X_3 = 1$, and $X_4 = 0$). In any of these two states, and whichever is the node affected by a transient pulse, we can always find among the remaining three nodes two consecutive nodes having the values 1 and 0. We call these two nodes the *hold nodes* and the two other nodes the *affected nodes*. For instance, in the 0 state, if the node struck by a particle is X_2, the hold nodes are the nodes X_4, X_1. In the same state, if the struck node is X_3, the hold nodes are again X_4, X_1, while if the struck node is X_1 or X_4, the hold nodes are X_2, X_3. We can find similarly the hold nodes for the 1 state of the DICE cell and for any struck node. In any of these situations, we can check easily that the states of the hold nodes either are not affected by the effects of the particle strike or, in the worst case, they are driven to the high-impedance state. Thus, the hold nodes preserve their correct values.

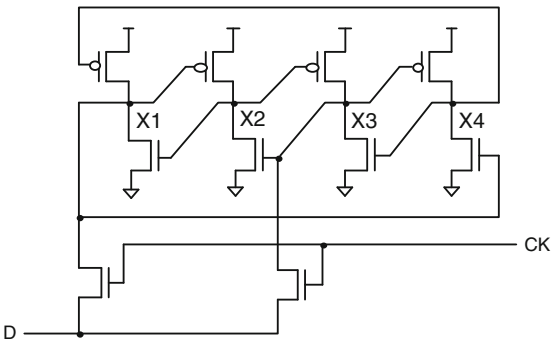


Fig. 8.9 A hardened storage cell

The state of the other two nodes can be modified by the particle strike, but the gate of one transistor of each inverter driving each of these nodes is controlled by one hold node. As a consequence, currents flowing through the transistors of these inverters quickly restore the correct values in the affected nodes.

By using the DICE cell, it is easy to built hardened SRAM cells, latches, and flip-flops. Thus, the DICE cell can be used to protect storage arrays like SRAMs, caches, register files, or CAMs; the configuration memory of FPGAs; and the latches or flip-flops in logic designs.

Comparison between a standard industrial flip-flop design and DICE-based flip-flop performed for a 90-nm process node shows 81% area overhead, identical rise time (0.13 ns in both cases), and a 20% fall time increase (0.12 vs. 0.10 ns). The 20 ps increase of the fall time will represent a small amount of the delay of a pipeline stage and will have a small effect on the clock frequency.

During static operation, the cell presents complete immunity for any transient affecting a single node of the cell and regardless of the transient strength. During dynamic operation (which is mostly of no concern for memory cells but cannot be neglected for latches and flip-flops), a transient affecting a single node of the cell could induce an erroneous state, but this can only happen if the cell is operating with stringent timing conditions, and even in this case, the particle strike should happen at specific time frames to induce an error. Its protection level can, however, be affected by multi-node upsets, which are gaining importance as technology moves deeper in the nanometric domain.

On the whole, the cell presents excellent hardening characteristics, but its practical use can be impacted by non-negligible extra area and power with respect to standard storage cells. Several other hardened cells can be found in the literature (e.g., [31, 33–35]). Their interest should be analyzed with respect to the goals and constraints of the target design. Some of them require lower area and/or power than DICE and can be preferred in certain situations. Some new hardened principles were proposed in [35] and used to design several hardened latch and memory cells. Figure 8.10 presents a fast hardened latch implementation from [35]. In [36], this design was adapted to implement a memory cell. The cell was shown to provide similar protection against SEUs as DICE, while requiring 20% less area and a 55% reduction of speed-power product.

8.6.2 SET Mitigation

Protecting combinational logic is more complex than protecting memories or latches and flip-flops, and may involve quite high hardware and power cost. It is, therefore, mandatory to select properly the protection scheme in order to meet design requirements in terms of area, speed, power, and reliability. Some approaches reduce SET sensitivity by using gate resizing to increase signal strength or node capacitance, [37–40]. These techniques may require moderate hardware cost. However, as they target only SETs, an additional solution is needed for SEUs,

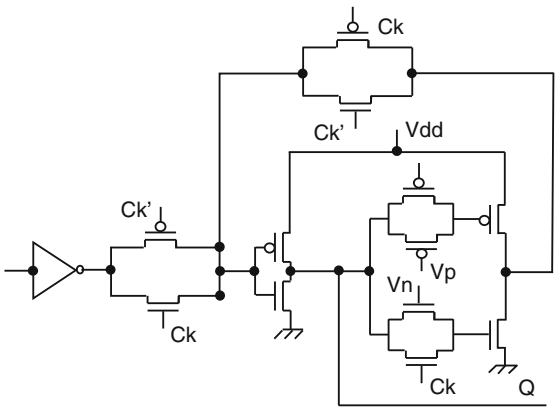


Fig. 8.10 A fast hardened latch cell

increasing hardware cost. Furthermore, resizing becomes increasingly inefficient as we move down to the nanometer scale.

Some other schemes allow mitigating both SEUs and SETs and are therefore more attractive. This chapter is mostly dedicated to these schemes.

Basically there are two approaches for protecting logic: error masking and error detection. A typical example of error masking is TMR, where the circuit under protection is triplicated and a majority voter determines the circuit output. A typical example of error detection is duplication and comparison. Excepting some systems in which the fail-stop concept can be used (the system operation is stopped upon error detection), in the majority of systems, error detection should be combined with a recovery scheme that brings the system into a correct state from which the operation can be resumed. Since soft errors are transient faults, the recovery could consist of reexecuting the latest operation (e.g., instruction replay). Another possibility consists on using checkpoints, that is, particular points of the system operation in which the system state is saved in a mass storage media. When an error is detected, the system operation resumes from the state saved during the last checkpoint (rollback recovery) [22, 41].

Because massive redundancy schemes such as TMR and duplication involve excessive hardware cost and power dissipation, in the following we discuss alternative schemes aimed at reducing this cost.

8.6.2.1 Time Redundancy Implemented in Software

These approaches are presented extensively in Chap. 9. Thus, we will briefly report some of them. A first software-based soft-error mitigation technique executes duplicated instructions over duplicated data and compares the results to check their integrity [42, 43]. This protection does not detect control-flow errors, which are often the most dangerous. To detect such errors, extra instructions are added in strategic points of the program to check its flow correctness [44]. Software rollback can also be implemented to achieve error recovery [45]. The interest of the

software-implemented approach is its flexibility, since it does not require any hardware modification. The counterpart is a high memory overhead (5× for error detection, 6× for error detection and recovery), high speed penalty (3× for error detection, 4× for error detection and recovery) [46], and high power penalty for executing a given task. The costs can be reduced if control-flow checking [44, 47, 48] alone is used, but the protection is incomplete. Evaluation of the cost of these approaches reported in [44] for four different benchmark programs shows a speed penalty varying from 107 to 185% for [48], from 120 to 426% for [47], and from 110 to 354% for [44]. The memory overhead varies, respectively, from 124 to 338% for [48], from 153 to 630% for [47], and from 129 to 496% for [44]. The significant differences between the costs incurred by these methods are reflected in important differences of their error detection efficiencies.

Multithreading is another software-based transient fault detection and recovery approach [49, 50] that reduces memory and speed cost by exploiting the capabilities of modern processors to execute multiple threads of computation. The performance penalty of this approach is only 40% [51], but it is less flexible than the first approach as it requires some hardware modifications.

The applicability of both approaches is limited to processor cores, so, for designs of different nature, only hardware-based approaches can be used.

8.6.2.2 Self-Checking Design

Self-checking (S-C) design is used to achieve concurrent error detection by means of hardware redundancy. A complex circuit is partitioned into its constituent functional blocks and each of these blocks is implemented according to the structure of Fig. 8.11. This structure implements functional blocks that deliver outputs belonging to an error detecting code, and thus introduces an invariant property that can be checked concurrently with circuit operation. A checker monitoring this code performs the concurrent error detection.

The desirable goal to be achieved by self-checking circuits is often stated as Totally Self-Checking (TSC) goal. This goal requires that under any modeled fault, the first erroneous output of the functional block is signaled on the outputs of the checker. To achieve this goal, some properties must be verified by the functional block and the checker. These properties are introduced by Carter [52] and formalized by Anderson [53].

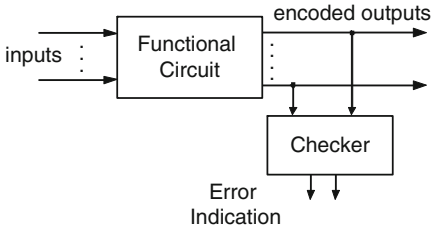


Fig. 8.11 General structure of self-checking circuits

Concerning the functional block, the following properties are required.

Fault Secure Property

A circuit is fault secure if under each modeled fault the produced erroneous outputs do not belong to the output code.

The reason for this property is obvious; if an erroneous output belongs to the code, the error is not detected and the TSC goal is lost. Thus, the fault secure property is the most important property required for the functional blocks. Another property used in the theory of self-checking systems is the self-testing one. This property stipulates that for each fault, there is at least one input vector, occurring during the circuit normal operation, that detects it. This property avoids the existence of redundant faults. Such faults remain undetectable forever and could be combined with new faults occurring later in the circuit, resulting in multiple faults that could destroy the fault secure property. Implementing this property requires removing redundancies during circuit design as such redundancies imply the existence of permanent faults that cannot be detected whatever are the stimuli applied on the circuit. As soft errors are not permanent faults, the self-testing property is less important in logic designs (errors disappear quickly as data in latches and flip-flops usually change in every clock cycle). In memories, scrubbing used to avoid accumulation of SEUs could be considered as a variant of this property.

The most obvious way for achieving the fault secure property is to duplicate the functional block and use a comparator to check for equality in the delivered outputs. Since this solution appeals for hardware and power penalties higher than 100%, more elaborated techniques are developed for reducing this cost. These techniques use error detection codes with cost lower than that for the duplication code. The most useful of these codes are as follows:

- *Parity code*. It detects all single errors and more generally all errors of odd multiplicity. It is the cheapest code since it adds only one check bit to the information bits. This check bit is computed to make the parity of each code word constant. We can use an odd parity code (odd number of 1s in each code word) or an even parity code (even number of 1s in each code word).
- *Dual-rail code*. It is a variety of the duplication code where the check bits are equal to the complements of the information bits. This code has very strong error detection capabilities since it detects any error affecting either the information part or its complement. But of course it is quite expensive since it duplicates the information part and usually the hardware.
- *Unordered codes*. In such codes, there are no two different code words x and z such that x covers z (to be written $x > z$), where x covers z means that x has a 1 in each bit position z has a 1. For instance, if $x = 10010$ and $z = 10001$, then neither x covers z nor z covers x and these words can both belong to an unordered code. On the contrary, if $x = 10010$ and $z = 10000$, then x covers z and these words cannot both belong to an unordered code. From this property, if an error

which includes only bit errors of the type $1 \rightarrow 0$ affects a code word x, then the resulting word z is covered by x, and z cannot belong to the unordered code. Thus, the error is detected. Similarly, if an error which includes only bit errors of the type $0 \rightarrow 1$ affects a code word x, the resulting word z will cover x and the error is again detectable. As a matter of fact, unordered codes detect all unidirectional errors. The most interesting unordered codes are the m-out-of-n code and the Berger code.

An m-out-of-n code [54] is a non-separable code (information and check bits are merged). It is composed of code words that have exactly m 1s, (e.g., 2-out-of-4 code: 1100, 1010, 1001, etc.). This code is an optimal non-separable unordered code (minimal redundancy for unordered coding).

The Berger code [55] is a separable (usually referred to as systematic) unordered code. The check part of this code represents the number of 0s in the information part. For instance, for an information part $I = 100101$, the check part will be $C = 011$. In the second variance of this code, the check part is equal to the complement of the number of 1s in the information part. For instance, for an information part $I = 100101$, the check part will be $C = 100$. This code is an optimal separable unordered code. For k information bits, the number of check bits is equal to $\lceil \log_2(k+1) \rceil$.

- *Arithmetic codes* [56, 57]. These codes are divided into separable and non-separable (systematic).

  In separable arithmetic codes of base A, the code words are obtained by associating to an information part X a check part X' equal to the modulo A of the information part, that is, $X' = |X|A$ (residue code), or $X' = A - |X|A$ (inverse residue code).

  In non-separable arithmetic codes of base A (or AN codes), the code words are equal to the product of the original (i.e., non-coded) word by the base A.

  Arithmetic codes are interesting for checking arithmetic operations, because they are preserved under such operations. The most useful arithmetic codes are the separable ones, and they are most often implemented as low-cost arithmetic codes [58], where the check base A is equal to $2^m - 1$. In this case, an m-bit modulo A adder can be realized as an m-bit adder having the carry-out signal feeding back the carry-in signal (carry end-around adder). Then, the check part generator for the low-cost arithmetic codes is realized as a modular network using these adders as building blocks. Low-cost arithmetic codes detect various types of arithmetic errors according to the value of the check base. For $A = 3$, corresponding to 2 check bits, all single arithmetic errors (i.e., when the arithmetic value of the error is a power of 2) are detected.

Design of Fault Secure Functional Blocks

The design of fault secure functional blocks is a difficult problem. For a given output code, one must guarantee that each modeled fault affecting a component of the circuit creates local errors that propagate to the circuit outputs as errors

detectable by the output code. The choice of the output code is a very critical task. Selecting a code with high error detection capabilities makes easier the achievement of the fault secure property but adds a large number of output signals, thus increasing hardware cost. On the contrary, selecting a code with low error detection capabilities will add less extra outputs but, for achieving fault secureness, it may require significant modifications of the circuit structure (and thus extra cost). As a matter of fact, the selection of the output code is made by taking into account the particular circuit structure, in order to obtain the best result.

Fault Secure Design Using Parity Codes

If a node of a circuit is connected with multiple circuit outputs, then a fault affecting this node may produce multiple output faults undetectable by the parity code. Thus, the parity code will be used in circuits where each internal node is connected with a single output (circuits with maximum divergent degree equal to 1). This is the case of circuits having a bit-slice structure in which there are no interactions between different slices. Such circuits naturally propagate internal errors due to single faults into single output errors, which are detected by the parity. Such blocks are for instance cell arrays of memories, register files, and buses. The parity code can still be efficient for circuits in which some logic is shared by several outputs. For instance, if only small parts of the logic are shared by several outputs, one can replicate these parts in a way that each replicate belongs to the cone of a single output. In other situations (e.g., PLAs), the circuit can be partitioned into sub-blocks such that each sub-block verifies the above structure [59].

A further possibility is to partition the circuit outputs into several groups such that any two outputs sharing some logic belong to different groups [60]. Then, a parity bit is used for each group, as shown in Fig. 8.12. In this figure, the functional outputs are partitioned into three groups: group 1 (S1, S4, S7, S10); group 2 (S2, S5, S8, S11); and group 3 (S3, S6, S9). No outputs belonging to the same group share some logic. Thus, we can achieve the fault secure property by generating a parity bit for each of these groups (P1 for group 1, P2 for group 2, and P3 for group 3). Attention has to be paid during the synthesis of the parity bits in order to ensure that

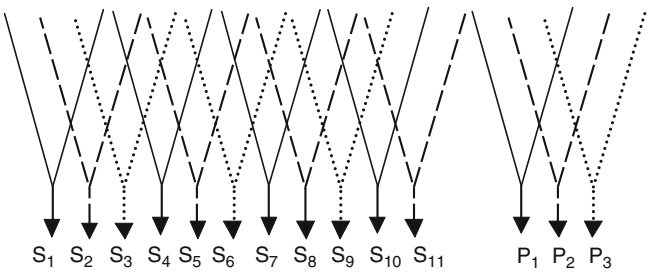


Fig. 8.12 Group parity coding

they do not share logic with their associated group of circuit outputs. Output partitioning and logic replication can also be combined [60, 61]. Experiments presented in [61] over 14 MCNC combinational benchmark circuits show an area overhead for the technique varying from 35 to 171%. Thus, this approach can be efficient for some blocks of a design and inefficient for some other blocks. For the later, the designer can select another scheme to protect them. In particular, this scheme is inefficient for arithmetic circuits, since the cones of any two outputs are overlapping.

The above approach is generic and can be used for any combinational functional block, but it often incurs high area penalty. Nevertheless, for some standard building blocks of VLSI circuits, such as adders, multipliers, dividers, and shifters, a thorough analysis may lead to specific fault secure solutions using parity codes. For instance, such solutions may constrain errors to propagate always on an odd number of outputs, may use specific detection means for some difficult faults, etc. Such techniques were used in [62–67], to protect various types of adders, ALUs, multipliers, and shifters. These solutions often achieve a low or moderate area cost (e.g., 15–24% for adders, 16–20% for shifters, and 40–50% for multipliers).

Fault Secure Design Using Unordered Codes

Unordered codes guarantee fault secureness in circuit structures where for each node of the circuit, all the paths between the node and the circuit outputs have the same inversion parity [68]. In this case, when a fault affecting a circuit node is propagated to the circuit outputs, it will be always inverted (if the inversion parity of the paths is 1) or always not inverted (inversion parity 0), resulting in unidirectional errors. This structure corresponds to unate functions. As a matter of fact, unordered codes can guarantee the fault secure property for a limited class of functions. Fortunately, it has been shown that if the inputs of a function belong to an unordered code, then the function can always be implemented as a unate one by exploiting the don't care input vectors [69]. Another possibility is to move all the inversions to the inputs of the circuit and implement the rest of the circuit as a unate function [59]. Thus, the errors produced in the inverter-free part are detected by any unordered code. On the other hand, the errors produced by faults in the primary inputs are checked by the input checker. This checker will be the checker of the previous functional block generating the input code of the current block, and does not require additional hardware overhead for checking the inputs of the current block. Fault secure solutions based on unordered codes have been proposed for various circuit structures including sequential circuits [70–72], PLAs [73], and ROMs [74]. Self-checking adders and ALUs based on unordered codes have also been proposed [75]. Another paper [76] presents a synthesis tool for fault secure finite state machines (FSMs) using unordered codes. However, unordered codes usually lead to higher area cost than the parity codes, principally due to the significantly higher complexity of their checkers.

Fault Secure Design Using Arithmetic Codes

Arithmetic codes are used to implement self-checking arithmetic operators. In modular arithmetic operators such as adders and multipliers built by using full and half adder cells, single faults in a cell produce single arithmetic errors on the cell outputs (error of the form 2^i). These errors are added to or subtracted from the final result, resulting in single arithmetic error on the outputs. Thus, using an arithmetic code of base $A = 3$ achieves fault secureness. In arithmetic operators using non-regular structures (e.g., using carry lookahead adders), a thorough analysis is required to determine the check base ensuring fault secureness [77–79]. Arithmetic codes are very efficient for protecting parallel multipliers, especially if the operant size is large. Experiments using the automation tool presented in [79] show a 10% area overhead for a 32 × 32 multiplier and a 6% area overhead for a 64 × 64 multiplier.

Fault Secure Designs Versus Transient Faults

The solutions described above detect both permanent and transient faults. However, these solutions were originally proposed to cover stuck-at faults and it was shown that the fault secure property holds for these faults. Various studies also show that the fault secure property holds also for other faults (like shorts [80]), under the condition that they produce logic errors. However, if the errors take intermediate values (i.e., analog values between the logic 0 and the logic 1 levels), or they a the timing characteristics of the circuit (like transient faults and timing faults), the fault secure property may not be valid. In this subsection, we are interested about SETs.

Stuck-at faults modify the value of the affected node during the whole duration of the clock cycle. Then, all the outputs connected to the affected node through sensitized paths are erroneous. On the contrary, an SET will produce an error on an output only if the transient pulse overlaps with the clock event when it reaches this output. Thus, an SET initiated from a circuit node may affect only a subset of the outputs affected by a stuck-at fault initiated from the same node. Therefore, the errors produced by transients are different than the errors produced by stuck-at faults. As a matter of fact, the detection of the later does not guarantee the detection of the former. Indeed, the fault secure property for transient faults is guaranteed only for a certain class of circuits designed to be fault secure for stuck-at faults [81]. These circuits are referred to as path-secure circuits. They achieve the fault secure property by means of structural constraints: the circuit structure is such that the propagation of an error from an internal node to the circuit outputs, through any collection of paths, gives errors detected by the output code. Among the fault secure designs described above, those checked by the parity code and using parity groups and those checked by unordered codes and using inversion-free functions are path secure. The other designs, that is, those using parity codes to implement adders, ALUs, multipliers, and shifters [62–67], and those using arithmetic codes to implement adders and multipliers [58, 77–79], are

not path secure. As the fault secure property for transient faults is not achieved for these designs, before selecting any of them, its efficiency should be evaluated by means of a fault injection tool considering circuit timing constraints, like the one described in [82].

8.6.2.3 Requirements and Implementation of Efficient Fault-Tolerant Logic

While concurrent error detection and fault tolerance are natural means for improving reliability and yield, they may incur significant area and power penalties. Self-checking techniques as the ones described in the previous section often incur high area and power penalties, and in certain cases (non-path-secure circuits), their efficiency against soft errors and other faults that gain significance in nanometric technologies can be compromised. The traditional fault-tolerant approaches based on duplication or TMR incur even higher area and power penalties (respectively, more than twice or three times the area and power of a non-protected design). Similarly, software-implemented time redundancy schemes, which repeat each operation and compare the results to detect temporary faults, multiply by almost three times power dissipation and have a similar impact on performance [45].

While massive (hardware or software) redundancy approaches, as the ones discussed above, are still necessary for safety critical applications and also for fail-safe systems [83], their large power and/or area and/or speed penalties make them undesirable for many applications. More importantly, as low power dissipation is a stringent requirement in advanced technologies, a breakthrough is required with respect to the traditional concurrent error detection and fault-tolerant approaches, in order to improve at the same time power, reliability, and yield in advanced nanometric technologies without excessive area and speed penalties.

Efficient Circuit-Level Time Redundancy

To achieve concurrent error detection or fault tolerance, the approaches discussed above either use hardware redundancy or replicate the operations of the circuit, leading to high power dissipation and high area or speed penalties. More recent schemes [84–86] avoid both hardware redundancy and operations redundancy. Instead, they detect or tolerate faults by observing the output signals of logic blocks at different instants.

A first scheme proposed in [84–86] observes the outputs of a combinational circuit at two different instants. This can be done by adding a redundant latch to each circuit output and driving the redundant latches by means of a delayed clock signal, as shown in Fig. 8.13a. The delayed clock signal could be produced by adding some delay elements on the normal clock signal. Another more efficient alternative is to use the two edges (rising and falling) and/or the two levels (high and low) of the clock signal to activate the regular and the redundant sampling element [87]. Because it samples the circuit outputs at two different instants, more

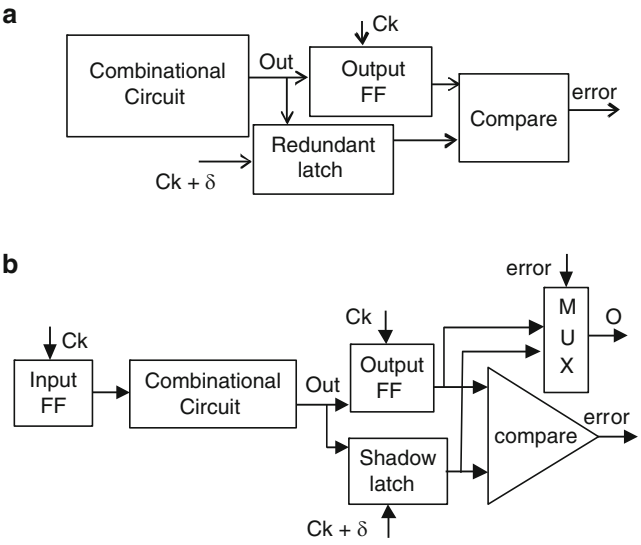


Fig. 8.13 (**a**) A detection technique using a redundant latch and a delayed clock [84, 85]. (**b**) The same technique extended in the RAZOR architecture [88, 89] to perform error correction

recently this scheme is also referred to as *double sampling*. Upon error detection, the latest operation has to be repeated (retry). Also, as suggested in [84, 87], to avoid the occurrence of the same error in case of timing faults, the clock frequency is reduced during retry. Furthermore, in case of frequent error detections, the clock frequency is permanently reduced [84]. The method was proposed in [84] to improve reliability (against SEUs, SETs, and timing faults) and/or to increase the operating frequency by detecting and correcting errors produced when a clock cycle shorter than the worst case circuit delay is used [84].

This scheme was later extended [88, 89] to introduce local error correction circuitry: upon error detection, the content of the redundant latch is used to replace the content of the functional flip-flop, as shown in Fig. 8.13b. This architecture was used by the University of Illinois and ARM to implement a dynamic voltage scaling approach in which supply voltage is reduced aggressively and the errors induced by this action are detected and corrected. The approach was shown to achieve 44% power dissipation reduction [90].

Another technique proposed in [84, 85] is shown in Fig. 8.14a. In this double-sampling scheme, a delay δ is added on the combinational circuit output rather than on the clock. This creates a second sampling instant that precedes by δ the regular sampling instant. The approach proposed in [91, 92] protects designs against accelerating aging affecting advanced nanometric process nodes, by monitoring the delay margins on the outputs of the combinational circuit. If the delay of the circuit is degraded due to aging, then the reduction of delay margins is detected before the occurrence of a timing fault and the clock frequency is reduced to

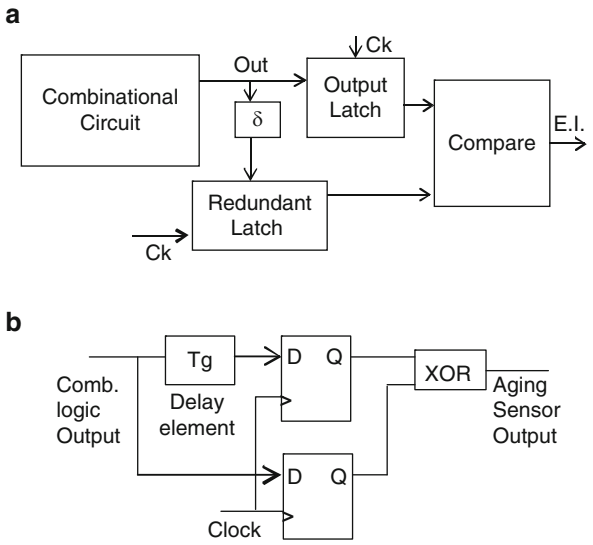


Fig. 8.14 (**a**) A detection technique using delayed signals [84]. (**b**) Use of the circuit of (**a**) to detect timing degradation before the occurrence of timing faults [91, 92]

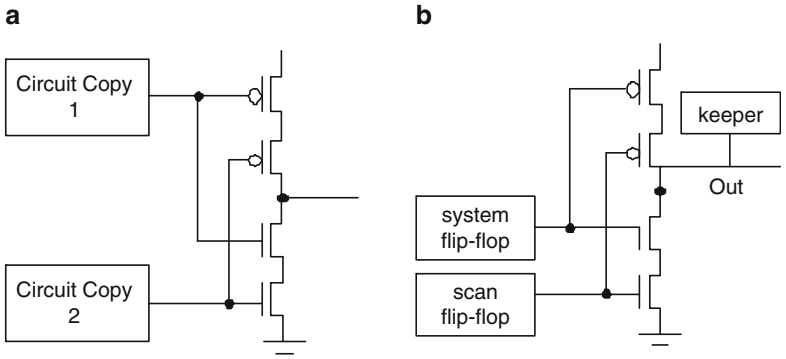


Fig. 8.15 (**a**) Fault-tolerant scheme for mitigating transient faults [84]. (**b**) The same principle adapted by Intel to protect flip-flops against SEUs [93]

maintain delay margins constant. This scheme is shown in Fig. 8.14b. The circuit in Fig. 8.14a [84] is used as a possible implementation of the aging detector.

Yet another technique proposed in [84] is shown in Fig. 8.15a. It uses an inverting gate composed of four transistors (known as *c*-element) in order to eliminate transient faults. The technique is later adapted in [93] to protect flip-flops against soft errors, as shown in Fig. 8.15b. The later technique exploits the existence of duplicated flip-flops in certain types of scan-chains, to reduce area cost.

In order to cope with both aging-induced delay faults and SEUs, Intel researchers combine the implementations presented in Figs. 8.14b and 8.15b, leading to the AVERA architecture [94].

While the idea introduced in [84] and the related schemes shown in Figs. 8.13–8.15, avoid massive redundancy, all these schemes require duplicating the system flip-flops. This duplication results in significant area and power penalties and makes necessary tradeoffs between protection level on the one hand and area and power penalties on the other hand (e.g., protecting only the subset of most sensitive flip-flops to avoid excessive area and power penalties). Furthermore, the schemes of Fig. 8.13a, b suffer from the following drawbacks:

- To avoid false alarms (and miscorrections in Fig. 8.13b), all path delays of the combinational circuits must exceed the delay δ of the delayed clock signal. This condition may require padding to increase the delay of short paths and may induce a significant cost. Also, as process variations worsen, it becomes increasingly difficult to guarantee that delays of short paths exceed the value δ in all circumstances. In addition, false alarms may be activated by clock skews, which may be even more difficult to fix.
- Due to a similar problem, we are obliged to use moderate values for the delay δ, limiting the duration of detectable delay faults.

Furthermore, the scheme in Fig. 8.13b corrects timing faults but not transient faults (SETs) and SEUs, neither latch/FF transition nor retention faults that may occur in low supply voltage modes. This is because these faults may alter indistinguishably the values of the regular flip-flops and the redundant latches. Thus, upon detection of an error that occurred in a redundant latch, the use of its content for performing error correction introduces errors in the functional flip-flops.

As for the scheme of Fig. 8.14b, as well as the combination of this scheme with the scheme of Fig. 8.15b in AVERA [94], they are designed to detect incremental delay increases developed slowly due to aging. Thus, they may not be efficient against large delay faults occurring due to variability, spot defects, and other causes. For the same reasons, these schemes may not be efficient against SETs. Finally, the schemes in Fig. 8.15 allow copying with soft errors but not with delay faults.

Hence a new fault-tolerant architecture resolving these problems is of great importance for addressing reliability, yield, and power issues in upcoming nanometric process nodes. The reason for which the above schemes could not cope with large delay faults, and also require adding a redundant latch, is that all the stages of a flip-flop-based design make computations at the same time and as soon as data are fetched in the flip-flops, a new computation cycle starts. The consequence is that the combinational circuit outputs are stable for short time durations, leaving a short time that we could exploit for error checking purposes.

A FF-based design is illustrated in Fig. 8.16a. By analyzing more carefully the operation of the flip-flops, we observe that when the master part of a flip-flop is transparent, the slave part is on memorization and vice versa. Thus, if we transform a flip-flop-based design (Fig. 8.16a) into its equivalent latch-based design

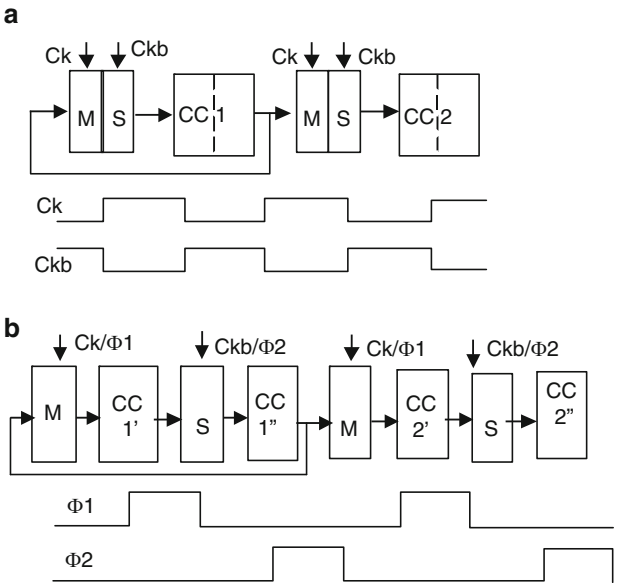


Fig. 8.16 (**a**) A FF-based design. (**b**) Its transformation into a latch-based design

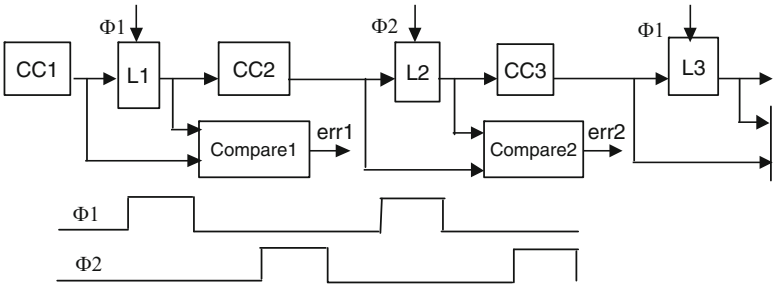


Fig. 8.17 Efficient error detection architecture

(Fig. 8.16b), by moving the slave latches from the outputs of the master latches to the middle of the combinational circuits as shown in Fig. 8.16b (where Ck and Ckb are replaced by non-overlapping clocks Φ_1 and Φ_2), we obtain a design that can operate at the same clock frequency as the original one. In addition, as observed above, the master and slave latches operate at different phases. This implies that in the modified design, the output signals of any combinational block (inputs of the latches) remain stable during the period in which its adjacent blocks are in computation. Thus, we can compare the outputs of the latches against their inputs (as proposed in the GRAAL architecture [95] and illustrated in Fig. 8.17), to detect delay faults of large duration. The scheme also detects SETs, SEUs, and all kinds of

latch faults including latch transition faults and latch retention faults. Therefore, this scheme demonstrates powerful error detection capabilities for all fault types of interest for nanometric designs (SEUs, SETs, delay faults, and latch faults).

Considering that the delay of each latch-based pipe-line stage will be roughly half of that of the FF-based pipe-line stage, the delays of each latch-based pipe-line stage will not exceed a half of those of the clock period (considering the same clock period for the FF-based and the latch-based case). In this case, the outputs of each pipe-line stage will be ready at the rising edge of the corresponding clock. Based on these assumptions, and using a more accurate estimation than that in [95], we can show that the scheme of Fig. 8.17 detects delay faults of duration up to 100% of the critical path delays (50% of the clock period T), without imposing any constraint on short paths. Thus, it offers very comfortable timing margins that can be employed for detecting large delay faults induced by spot defects, variability, and circuit aging, and at the same time reduce power dissipation by means of the idea introduced in [90] (i.e., reduce Vdd level and detect and correct the induced timing errors). If we add short path constraints, the detectable delay faults are further increased. For instance, we can impose short paths to have delays larger than the high level of the clock signals. Then, we can detect timing faults of duration much higher than the circuit delays (up to $T/2 + T_h$, where T_h is the duration of the high level of the clock). In the typical case, $T_h = T/4$, this will result in the detection of delays up to 150% of the clock cycle. We can also find that all transient faults (SETs) of duration not exceeding $T/2 - T_h$ are detected by the scheme that does not use short path constraints, while the scheme using short path constraints detects all transient faults of duration not exceeding $T/2$.

The circuit of Fig. 8.17 also tolerates clock skews of up to T_h, as an inherent property of latch-based designs [96]. In addition, a thorough analysis shows that, thanks to the detection capabilities of the architecture of Fig. 8.17, larger clock skews that are not tolerated are always detected.

While these properties make the proposed architecture very robust to clock skews, in complex SoCs composed of several tens or hundreds of processing nodes, global clock skews may become larger than the tolerant skew duration, thus inducing frequent error detections. For such complex designs the proposed scheme may have to be combined with a globally asynchronous, locally synchronous architecture (GALS) approach. In other designs, a globally single-clock locally double-clock architecture could be used.

The individual error detection signals are compacted by means of OR trees into a global error detection signal for each odd and each even pipe-line stage. A single global error detection signal for all odd and for all even pipe-line stages could also be generated. In the case where no short path constraints are enforced, the global error detection signal of each odd/even stage is latched at the rising edge of Φ_2/Φ_1. In the case where short path constraints are enforced (the delay of any path exceeds the time duration of the high level of the clock signals), the global error detection signal of each odd/even stage is latched at the falling edge of Φ_2/Φ_1. The actual checking of a pipe-line stage will be accomplished at an instant preceding the rising or the falling edge of Φ_2/Φ_1 by a time equal to the delay of the corresponding OR

tree. Delays can be added on Φ_2/Φ_1 signals used to latch the global error signals if the designer desires to compensate this effect.

The implementation employing short path constraints offers a higher protection not only for timing faults (as we have already discussed), but also for soft errors. An SEU may occur in a latch at any time within the clock cycle. If the odd/even global error signals are latched at the rising edge of Φ_2/Φ_1 (which is the case if no short-path constraints are enforced), an SEU occurring after this edge in an odd/even latch will not be detected. Thus, since the delays of short paths are not guaranteed to be larger than the high-level duration of the clock signals, the error may have time to propagate until the latches of this stage before the falling edge of Φ_2/Φ_1, resulting in undetectable error. The probability of this situation depends not only on the delays of short paths, but also on the delays of the OR trees, if these delays are not compensated as suggested in the previous paragraph. This situation will not happen in the case where short path constraints are enforced, guarantying detection of any harmful SEU.

Concerning speed, some of the comfortable timing margins of the error detection described above could also be used to trade error detection capabilities with speed (e.g., sacrifice a part of these margins to increase speed by means of time borrowing).

Last but not least, thanks to the long duration of signal stability in latch-based designs, error detection in Fig. 8.17 does not duplicate the system latches. Only comparators are added to compare the inputs of latches against their outputs. As a matter of fact, the scheme of Fig. 8.17 is expected to induce lower area and most importantly drastically lower power penalties with respect to the schemes of Figs. 8.13–8.15, which introduce extra clocked cells (latches) that are gourmand in power. As power penalty is one of the stronger constraints in nanometric designs, it makes inacceptable the use of these schemes for protecting all the signals of the circuit, which is necessary for SEUs and SETs.

A problem related with the scheme of Fig. 8.17, as well as with the schemes of Fig. 8.13a, b, is the occurrence of metastability. Metastability can lead to undetected errors as metastable values could be interpreted differently by the next pipeline stage and by the error detection circuitry. Thus, it may reduce the mean time to undetected failure. According to the mean time to failure and the target mean time to undetected failure, metastability detectors, as the ones described in [88, 89], may be required.

Note that Fig. 8.17 achieves error detection only. Error correction can also be implemented locally. However, this scheme will require duplicating the circuit latches and will induce significant area and power costs. Instruction replay schemes, as the one implemented in the IBM power6 processor [97], are much more efficient for this purpose.

The double-sampling approach of Fig. 8.13 can also be used to protect interconnections. This is straightforward, as interconnections can be seen as combinational circuits implementing an identity function, whose delay may be affected by delay faults due to variability, aging, and most importantly crosstalk. Figure 8.18a shows this implementation. The shadow latches are rated by Ckb. If we consider a

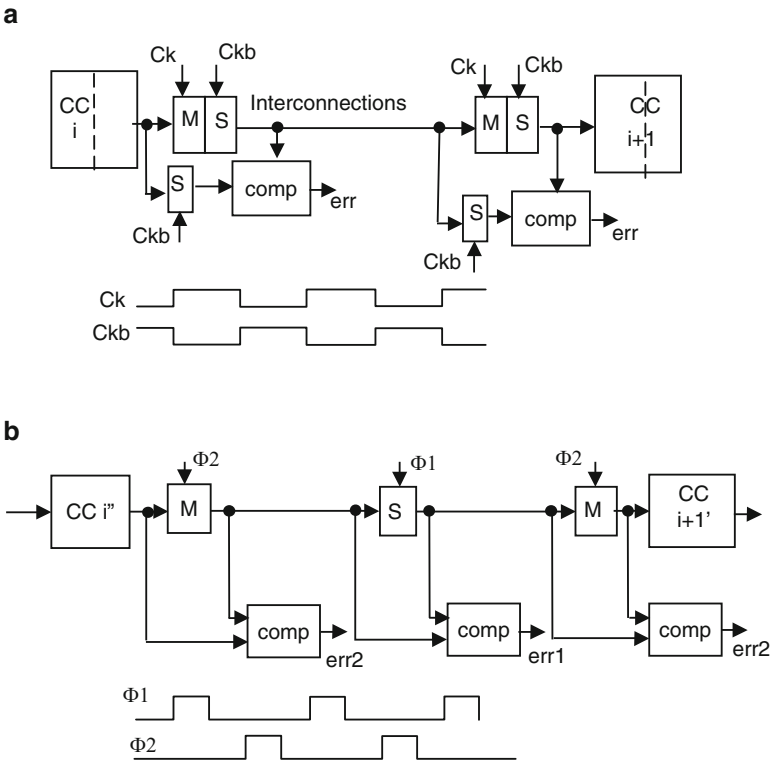


Fig. 8.18 (**a**) Flip-flop-based and (**b**) GRAAL-based error detection in interconnections

50% clock duty cycle, the scheme detects delay faults that increase the circuit delays by a half clock period. The short paths constraint is not a problem as far as the min delays of the interconnections (i.e., in the favorable crosstalk conditions where neighbor signals make transitions in the same direction) are not shorter than half clock period. This condition is not difficult to achieve unless we use quite slow clocks. But it may be a problem in a complex circuit where some interconnections are shorter than the other, or if we increase the clock period when some parts of the circuit are affected by delay faults. It the later case, we have to use a clock generation circuit that does not increase the duration of the high level of the clock when it increases the clock period.

The GRAAL approach can be used for the same purpose, as shown in Fig. 8.18b. The interconnections are divided in two parts of equal lengths by inserting a latch at half length of the interconnections. For latches clocked by Φ_1, the error signals can be observed at the rising edge or the falling edge of Φ_2, while for latches clocked by Φ_2, the error signals can be observed at the rising edge or the falling edge of Φ_1. Using the rising edge allows detecting delay faults of duration up to the half of the clock period, which is larger than 100% of the delays of the checked interconnection parts (half lengths), while using the falling edge of the clock allows detecting

delay faults of duration up to 75% of the clock period, which is larger than 150% of the delays of the checked interconnection parts. The constraint for using the later approach is that the min delays of the divided interconnection parts (i.e., in the favorable crosstalk conditions where neighbor signals make transitions in the same direction) are not shorter than 25% of the clock period. This will not be difficult to achieve unless we use quite slow clocks. We observe that without introducing any short delay constraints, the GRAAL approach detects twice larger delay faults (as percentage of the interconnection delays) than the flip-flop-based double-sampling scheme. It detects three times larger delay faults when it introduces equivalent short delay constraints as the flip-flop-based double-sampling scheme. In particular, GRAAL achieves these excellent error detection capabilities at negligible (a few percent) area and power penalties.

Recently, improvements of the scheme given in Fig. 8.13 were proposed and evaluated by Intel [98]. The proposed solutions referred to as transition detection time borrowing (TDTB) and double-sampling time borrowing (DSTB) exploit the fact that in the double-sampling approach of Fig. 8.13a, b, all path delays exceed the time duration of the high clock level. Thus, the data on the combinational block outputs are stable until the falling edge of the clock. As a consequence, we do not need to latch these data on the rising edge of the clock and the master latch of the output flip-flop can be eliminated. This is shown in Fig. 8.19a, b, where a transparent latch is connected to the combinational block output. This latch samples the data present on this output during the rising edge of the clock. Furthermore, for any timing fault whose extra delay does not exceed the time duration of the high clock level, there is no risk of metastability on the latch output. Metastability is possible on the detection signal's path, but the probability that an erroneous output goes undetected due to this problem is relatively low [98].

Two kinds of detection circuits were used. The first (Fig. 8.19a) is a transition detector [99, 100], which detects any signal transition occurring during the high clock level. The second (Fig. 8.19b) is a shadow flip-flop used to implement a double-sampling scheme as proposed in [84, 85]. The flip-flop samples data at the rising edge of the clock and the latch at the falling edge of the clock. Comparison of the flip-flop and latch outputs detects timing faults whose delay does not exceed the time duration of the high clock level. The scheme in Fig. 8.19a induces lower power

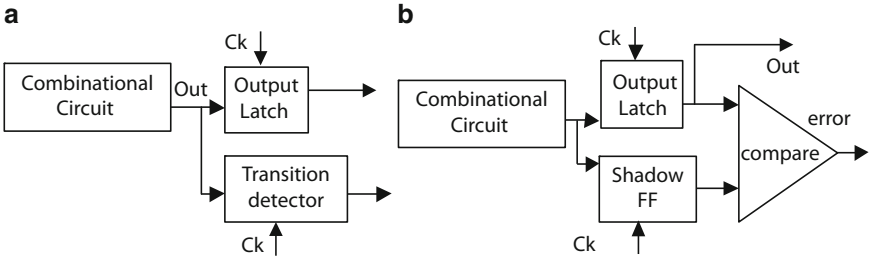


Fig. 8.19 (**a**) Transition detection time borrowing (TDTB) scheme [98]. (**b**) Double-sampling time borrowing (DSTB) scheme [98]

dissipation than the scheme in Fig. 8.19b (the transition detector consumes less power than the shadow flip-flop). Figure 8.19a also consumes less power than a traditional flip-flop design, as the transition detector and the latch dissipate less power than a flip-flop.

The drawbacks of the scheme in Fig. 8.19a are a more difficult design with respect to Fig. 8.19b (in particular, under variability and aging issues) and a protection of the output latch against SEUs during a limited period of the clock cycle. The scheme given in Fig. 8.19b can protect the output latch against SEUs during the entire clock cycle if the comparator output is validated at the end of this cycle.

Comparing TDTB and DSTB schemes with GRAAL, we find that these schemes are compatible with more traditional design, while GRAAL is used with non-overlapping clocks. Also, TDTB could dissipate less power than GRAAL. In fact, GRAAL uses an output latch and an XOR gate in each pipe-line stage, while TDTB uses an output latch and a transition detector. The transition detector will dissipate more power than the XOR gate, in particular as it loads the clock tree. However, a latch-based architecture, as the one used in GRAAL, will have more pipe-line stages to achieve the same speed as a FF-based architecture.

Compared with DSTB, GRAAL should dissipate less due to the shadow FF added in the DSTB scheme. Also, GRAAL detects timing faults up to 100% of the pipe-line stage delay, which represents a much larger percentage of circuit delay with respect to the faults detected by TDTB and DSTB. Most importantly, in this case, we do not need to enforce short path constraints as is required in TDTB and DSTB. Such constraints further increase area and power, and may be difficult to enforce due to variability and clock skews. Finally, if we impose short path constraints, the detection capabilities of GRAAL can reach 150% of the delay of each pipe-line stage. On the contrary, GRAAL will exhibit data-path metastability. But this will happen if metastable values occur during the falling edge of the clock. This will correspond to timing faults of duration at least equal to the high level of the clock cycle. This represents a percentage of the pipe-line stage delay much higher than the faults detected by TDTB and DSTB (equal to the high level of the clock cycle too, but representing a smaller percentage of the pipe-line stage delay). In addition, the metastability issue will affect only partially the detection capabilities of GRAAL for faults of larger duration. A decision to use metastability detectors can be taken following an analysis of the intrinsic FIT of the design, the FIT improvements achieved by GRAAL without using metastability detectors, and the target FIT.

8.7 Conclusions

As soft errors become a stringent reliability issue in nanometric technologies, design for soft-error mitigation is gaining importance. As for any fault-tolerant approach, soft-error mitigation may significantly impact area, speed, and power, so the selection of the best approach is a complex tradeoff between these parameters

and the target level of reliability. It is, therefore, suitable to dispose a variety of solutions that can meet various constraints of the design, such as minimal area, minimum power, minimum speed, maximum reliability, or a trade-off between these parameters. We presented various schemes that exhibit such characteristics and could help the designer to meet product requirements. Active work in this area in various research centers should enrich the scope of solutions and improve our ability to face the soft-error and other reliability threats that accompany nanometric scaling.

References

1. Baumann R.C., Soft errors in advanced computer systems. IEEE Des. Test Comput. 22(3), 2005, 258–266
2. Baumann R.C., Hossain T.Z., Murata S., Kitagwava H., Boron compounds as a dominant source of alpha particles in semiconductor devices. Proceedings of the 33rd International Reliability Physics Symposium, 1995
3. Baze M., Buchner S., Attenuation of single event induced pulses in CMOS combinational logic. IEEE Trans. Nucl. Sci. 44(6), 1997, 2217–2223
4. Buchner S., Baze M., Brown D., McMorrow D., Melinger J., Comparison of error rates in combinational and sequential logic. IEEE Trans. Nucl. Sci. 44(6), 1997, 2209–2216
5. Benedetto J., Eaton P., Avery K., Mavis D., Turflinger T., Dodd P., Vizkelethyd G., Heavy ion-induced digital single-event transients in deep submicron processes. IEEE Trans. Nucl. Sci. 51, 2004, 3480–3485
6. Nicolaidis M., Scaling deeper to submicron: on-line testing to the rescue. Proceedings of the 28th Symposium on Fault-Tolerant Computing (FTCS-28), Munich, June 1998, pp. 299–301
7. Nicolaidis M., Design for soft-error robustness to rescue deep submicron scaling. Proceedings of the IEEE International Test Conference, Washington, DC, October 18–23, 1998
8. Nicolaidis M., Scaling deeper to submicron: on-line testing to the rescue. Proceedings of the 1998 IEEE International Test Conference (ITC 98), Washington, DC, October 1998
9. Levendel I., Nicolaidis M., Abraham J.A., Abramovici M., Motto S., A D&T roundtable: on-line test. IEEE Des. Test Comput. 16(1), 1999, 80–86
10. Hawkins C.F., Baker K., Butler K., Figueras J., Nicolaidis M., Rao V., Roy R., Welsher T., IC reliability and test: what will deep submicron bring? IEEE Des. Test Comput. 16(2), 1999, 84–91
11. Roche P., Jacquet F., Caillat C., Schoellkopf J.P., An alpha immune and ultra low neutron SER high density SRAM. Proceedings of IRPS 2004, April 2004, pp. 671–672
12. Nicolaidis M., Achouri N., Boutobza S., Dynamic data-bit memory built-in self-repair. Proceedings of the International Conference on Computer-Aided Design (ICCAD'03), San Jose, CA, November 9–13, 2003
13. Hsiao M.Y., A class of optimal minimum odd-weight-column SEC-DED codes. IBM J. Res. Dev. 14(4), 1970, 395–401
14. Richter M., Oberlaender K., Goessel M., New linear SEC-DED codes with reduced triple bit error miscorrection probability. 14th IEEE International On-Line Testing Symposium, IOLTS'08, July 7–9, 2008, pp. 37–42
15. Nicolaidis M., Electronic circuit assembly comprising at least one memory with error correcting means. French and US Patent pending, filed July 2001
16. Nicolaidis M., Data storage method with error correction. French and US Patent pending, filed July 2002
17. Nicolaidis M., Design for Soft-Error Mitigation. IEEE Trans. Device Mater. Reliab. 5(3), 2005, 405–418

18. Vargas F., Nicolaidis M., SEU tolerant SRAM design based on current monitoring. Proceedings of the 24th IEEE International Symposium on Fault-Tolerant Computing, Austin, TX, June 1994, pp. 106–115
19. Gill B., Nicolaidis M., Wolff F., Papachristou C., Garverick S., An efficient BICS design for SEUs detection and correction in semiconductor memories. Proceedings of the Conference on Design, Automation and Test in Europe (DATE), 2005, pp. 592–597
20. Reed I.S., Solomon G., Polynomial codes over certain finite fields. SIAM J. Appl. Math. 8, 1960, 300–304
21. Blahut R.E., Theory and practice of error control codes. Addison-Wesley, Reading, MA, 1983
22. Pradhan D.K., Fault-tolerant computing system design. Prentice-Hall, Englewood Cliffs, NJ, 1996
23. Peterson W.W., Weldon E.J., Error-correcting codes, 2nd edn. MIT, Cambridge, MA, 1972
24. Koopman P., Chakravarty T., Cyclic redundancy code (CRC) polynomial selection for embedded networks. International Conference on Dependable Systems and Networks (DSN-2004), Florence, Italy, June 28–July 1, 2004, pp. 145–154
25. Sklar B., Digital communications: fundamentals and applications. Prentice-Hall, Englewood Cliffs, NJ, 2001
26. Nicolaidis M., A low-cost single-event latchup mitigation scheme. Proceedings of the 12th IEEE International Symposium on On-Line Testing, Como, Italy, July 10–12, 2006
27. Nicolaidis M., Torki K., Natali F., Belhaddad F., Alexandrescu D., Implementation and validation of a low-cost single-event latchup mitigation scheme. IEEE Workshop on Silicon Errors in Logic – System Effects (SELSE), Stanford, CA, March 24–25, 2009
28. Nicolaidis M., Circuit intégré protégé contre les courts-circuits et les erreurs de fonctionnement suite au passage d'une radiation ionisante. French Patent 2,882,601, delivered September 21, 2007
29. Liu M.S., Shaw G.A., Yue J., Fabrication of stabilized polysilicon resistors for SEU control. US Patent, issued May 18, 1993
30. Rockett L., An SEU hardened CMOS data latch design. IEEE Trans. Nucl. Sci. NS-35(6), 1988, 1682–1687
31. Whitaker S., Canaris J., Liu K., SEU hardened memory cells for a CCSDS Reed Solomon encoder. IEEE Trans. Nucl. Sci. NS-38(6), 1991, 1471–1477
32. Calin T., Nicolaidis M., Velazco R., Upset hardened memory design for submicron CMOS technology. 33rd International Nuclear and Space Radiation Effects Conference, Indian Wells, CA, July 1996
33. Bessot D., Velazco R., Design of SEU-hardened CMOS memory cells: the HIT cell. Proceedings of the 1994 RADECS Conference, 1994, pp. 563–570
34. Omana M., Rossi D., Metra C., Novel transient fault hardened static latch. Proceedings of the 18th International Test Conference, 2003, pp. 886–892
35. Nicolaidis M., Perez R., Alexandrescu D., Low-cost highly-robust hardened storage cells using blocking feedback transistors. Proceedings of the IEEE VLSI Test Symposium, 2008, pp. 371–376
36. Lin S., Kim Y.B., Lombardi F., A 11-transistor nanoscale CMOS memory cell for hardening to soft errors. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2010
37. Moharam K., Touba N., Cost-effective approach for reducing the soft error failure rate in logic circuits. Proceedings of the International Test Conference (ITC), September 2003, pp. 893–901
38. Dhillon Y.S., Diril A.U., Chatterjee A., Soft-error tolerance analysis and optimization of nanometer circuits. Proceedings of the Conference on Design, Automation and Test in Europe (DATE), March 7–11, 2005, pp. 288–293
39. Zhou Q., Mohanram K., Cost-effective radiation hardening technique for combinational logic. Proceedings of the 2004 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), November 7–11, 2004, pp. 100–106

40. Dhillon Y.S., Diril A.U., Chatterjee A., Metra C., Load and logic co-optimization for design of soft-error resistant nanometer CMOS circuits. Proceedings of the 11th IEEE International On-Line Testing Symposium (IOLTS), July 6–8, 2005, pp. 35–40
41. Siewiorek D.P., Swarz R.S., Reliable computer systems: design and evaluation. Digital Press, Bedford, MA, 1992
42. Cheynet P., Nicolescu B., Velazco R., Rebaudengo M., Sonza Reorda M., Violante M., Experimentally evaluating an automatic approach for generating safety-critical software with respect to transient errors. IEEE Trans. Nucl. Sci. 47(6), 2000, 2231–2236
43. Oh N., Mitra S., McCluskey E.J., ED4I: error detection by diverse data and duplicated instructions. IEEE Trans. Comput. 51(2), 2002, 180–199
44. Goloubeva O., Rebaudengo M., Sonza Reorda M., Violante M., Soft-error detection using control flow assertions. Proceedings of the 18th International Symposium on Defect and Fault Tolerance in VLSI Systems, Cambridge, MA, November 3–5, 2003, pp. 581–588
45. Rebaudengo M., Sonza Reorda M., Violante M., A new approach to software-implemented fault tolerance. J. Electron. Test.: Theory Appl. 20, 2004, 433–437
46. Rebaudengo M., Sonza Reorda M., Violante M., Nicolescu B., Velazco R., Coping with SEUs/SETs in microprocessors by means of low-cost solutions: a comparative study. IEEE Trans. Nucl. Sci. 49(3), 2002, 1491–1495
47. Alkhalifa Z., Nair V.S.S., Krishnamurthy N., Abraham J.A., Design and evaluation of system-level checks for on-line control flow error detection. IEEE Trans. Parallel Distrib. Syst. 10(6), 1999, 627–641
48. Oh N., Shirvani P.P., McCluskey E.J., Control-flow checking by software signatures. IEEE Trans. Reliab. 51(2), 2002, 111–122
49. Mukherjee S., Reinhardt S., Transient fault detection via simultaneous multithreading. International Symposium on Computer Architecture, June 2000
50. Vijaykumar P.N., Pomerantz I., et al., Transient fault recovery using simultaneous multithreading. International Symposium on Computer Architecture, May 2002
51. Mukherjee S., Kontz S., Reinhardt S., Detailed design and evaluation of redundant multithreading alternatives. International Symposium on Computer Architecture, May 2002
52. Carter W.C., Schneider P.R., Design of dynamically checked computers. Proceedings of the 4th Congress IFIP, Vol. 2, Edinburgh, Scotland, August 5–10, 1968, pp. 878–883
53. Anderson D.A., Design of self-checking digital networks using coding techniques. Coordinated Sciences Laboratory, Report R/527. University of Illinois, Urbana, IL, September 1971
54. Freiman C.V., Optimal error detection codes for completely asymmetric binary channels. Inform. Contr. 5, 1962, 64–71
55. Berger J.M., A note on error detection codes for asymmetric binary channels. Inform. Contr. 4, 1961, 68–73
56. Peterson W.W., On checking an adder. IBM J. Res. Dev. 2, 1958, 166–168
57. Avizienis A., Arithmetic algorithms for error-coded operands. IEEE Trans. Comput. C-22(6), 1973, 567–572
58. Rao T.R.N., Error coding for arithmetic processors. Academic, New York, 1974
59. Nicolaidis M., Courtois B., Self-checking logic arrays. In: Microprocessors and microsystems. Butterworth Scientific Ltd, Guildford, UK, 1989
60. De K., Natarajan C., Nair D., Banerjee P., RSYN: a system for automated synthesis of reliable multilevel circuits. IEEE Trans. VLSI Syst. 2(2), 1994, 186–195
61. Touba N.A., McCluskey E.J., Logic synthesis techniques for reduced area implementation of multilevel circuits with concurrent error detection. Proceedings of the International Conference on Computer-Aided Design, 1994
62. Nicolaidis M., Duarte R.O., Manich S., Figueras J., Achieving fault secureness in parity prediction arithmetic operators. IEEE Des. Test Comput. 14(2), 1997, 60–71
63. Nicolaidis M., Duarte R.O., Design of fault-secure parity-prediction booth multipliers. IEEE Des. Test Comput. 16(3), 1999, 90–101

64. Duarte R.O., Nicolaidis M., Bederr H., Zorian Y., Efficient fault-secure shifter design. J. Electron. Test.: Theory Appl. 12, 1998, 29–39
65. Nicolaidis M., Carry checking/parity prediction adders and ALUs. IEEE Trans. VLSI Syst. 11(1), 2003, 121–128
66. Ocheretnij V., Goessel M., Sogomonyan E.S., Marienfeld D., A modulo *p* checked self-checking carry-select adder. 9th International On-Line Testing Symposium, July 2003
67. Ocheretnij V., Marienfeld D., Sogomonyan E.S., Gossel M., Self-checking code-disjoint carry-select adder with low area overhead by use of add1-circuits. 10th IEEE International On-Line Testing Symposium, Madeira, Portugal, July 2004
68. Smith J.E, Metze G., The design of totally self-checking combinatorials circuits. Proceedings of the 7th Fault-Tolerant Computing Symposium, Los Angeles, CA, June 1997
69. Mago G., Monotone functions in sequential circuits. IEEE Trans. Comput. C-22(10), 1973, 928–933
70. Diaz M., Design of totally self-checking and fail-safe sequential machines. Proceedings of the 4th International Fault-Tolerant Computing Symposium, Urbana, IL, 1974
71. Diaz M., Geffroy J.C., Courvoisier M., On-set realization of fail-safe sequential machines. IEEE Trans. Comput. C-23, 1974, 133–138
72. Nanya T., Kawamura T., A note on strongly fault secure sequential circuits. IEEE Trans. Comput. C-36, 1987, 1121–1123
73. Mak G.P., Abraham J.A., Davindson E.S., The design of PLAs with concurrent error detection. Proceedings of the FTCS-12, Santa Monica, CA, June 1982
74. Fuchs W.K., Chien Ch-H., Abraham J., Concurrent error detection in highly structured logic arrays. IEEE J. Solid-State Circuits SC-22, 1987, 583–594
75. Lo J.C., Thanawastien S., Rao T.R.N., Nicolaidis M., An SFS Berger check prediction ALU and its application to self-checking processors designs. IEEE Trans. Comput. Aided Des. 2(4), 1992, 525–540
76. Jha N.K., Wang S.-J., Design and synthesis of self-checking VLSI circuits. IEEE Trans. Comput. Aided Des. 12, 1993, 878–887
77. Sparmann U., On the check base selection problem for fast adders. Proceedings of the 11th VLSI Test Symposium, Atlantic City, NJ, April 1993
78. Sparmann U., Reddy S.M., On the effectiveness of residue code checking for parallel two's complement multipliers. Proceedings of the 24th Fault-Tolerant Computing Symposium, Austin, TX, June 1994
79. Alzaher Noufal I., A tool for automatic generation of self-checking multipliers based on residue arithmetic codes. Proceedings of the 1999 Design, Automation and Test in Europe Conference, Munich, March 1999
80. Nicolaidis M., Shorts in self-checking circuits. J. Electron. Test.: Theory Appl. 1(4), 1991, 257–273
81. Anghel L., Nicolaidis M., Alzaher Noufal I., Self-checking circuits versus realistic faults in very deep submicron. Proceedings of the 18th IEEE VLSI Test Symposium, Montreal, Canada, April 2000
82. Alexandrescu D., Anghel L., Nicolaidis M., New methods for evaluating the impact of single event transients in VDSM ICs. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, November 2002
83. Nicolaidis M., Fail-safe interfaces for VLSI: theoretical foundations and implementation. IEEE Trans. Comput. 47(1), 1998, 62–77
84. Nicolaidis M., Time redundancy based soft-error tolerant circuits to rescue very deep submicron. Proceedings of the 17th IEEE VLSI Test Symposium, Dana Point, CA, April 1999
85. Anghel L., Nicolaidis M., Cost reduction and evaluation of a temporary faults detecting technique. Proceedings of the Design, Automation and Test in Europe Conference (DATE), Paris, March 2000

86. Anghel L., Nicolaidis M., Cost reduction and evaluation of a temporary faults detecting technique. Chapter in The Most Influential Papers of 10 Years DATE, Springer Editions, ISBN: 978-1-4020-6487-6, 2008
87. Nicolaidis M., Circuit Logique protégé contre des perturbations transitoires. French Patent application, filed March 9, 1999
88. Ernst D., et al., Razor: a low-power pipeline based on circuit-level timing speculation. Proceedings of the 36th International Symposium on Microarchitecture, December 2003
89. Ernst D., et al., Razor: circuit-level correction of timing errors for low-power operation. IEEE Micro 24(6), 2003, 10–20
90. Das S., et al., A self-tuning DVS processor using delay-error detection and correction. IEEE Symposium on VLSI Circuits, June 2005
91. Agarwal M., Paul B.C., Ming Zhang Mitra S., Circuit failure prediction and its application to transistor aging. Proceedings of the 5th IEEE VLSI Test Symposium, Berkeley, CA, May 6–10, 2007
92. Mitra S., Agarwal M., Circuit failure prediction to overcome scaled CMOS reliability challenges. IEEE International Test Conference, Santa Clara, CA, October 23–25, 2007
93. Mitra S., Zhang M., Mak T.M., Kim K.S., System and shadow circuits with output joining circuit. US Patent 7,278,074, October 2, 2007, Assignee Intel
94. Ming Zhang Mak T., Tschanz J., Kee Sup Kim Seifert N., Lu D., Design for resilience to soft errors and variations. 13th IEEE International On-Line Testing Symposium (IOLTS 07), July 2007, pp. 23–28
95. Nicolaidis M., GRAAL: a new fault-tolerant design paradigm for mitigating the flaws of deep-nanometric technologies. Proceedings of the IEEE International Test Conference (ITC), Santa Clara, CA, October 23–25, 2007
96. Piguet C., et al., Low-power design of 8-b embedded CoolRisc microcontroller cores. IEEE J. Solid-State Circuits 32(7), 1997, 1067–1078
97. Mack M.J., Sauer W.M., Mealey B.G., IBM POWER6 reliability. IBM J. Res. Dev. 51(6), 2007, 763–764
98. Bowman K.A., et al., Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance. IEEE J. Solid-State Circuits 44(1), 2009, 49–63
99. Franco P., McCluskey E.J., On-line testing of digital circuits. Proceedings of the IEEE VLSI Test Symposium, April 1994, pp. 167–173
100. Metra C., Favalli M., Ricco B., On-line detection of logic errors due to crosstalk, delay, and transient faults. Proceedings of the International Test Conference, Washington, DC, October 18–23, 1998, pp. 524–533
101. Hamming R.W., Error detecting and error correcting codes. Bell Syst. Tech. J. 29, 1953, 147–160
102. Bose R.C., Ray-Chaundhuri D.K., On a class of error correcting binary group codes. Inform. Contr. 3(1), 1960, 68–79
103. Hocquenghem A., Codes Corecteurs d'Erreurs. Chiffres 2, 1959, 147–156

Chapter 9
Software-Level Soft-Error Mitigation Techniques

Maurizio Rebaudengo, Matteo Sonza Reorda, and Massimo Violante

Several application domains exist, where the effects of Soft Errors on processor-based systems cannot be faced by acting on the hardware (either by changing the technology, or the components, or the architecture, or whatever else). In these cases, an attractive solution lies in just modifying the software: the ability to detect and possibly correct errors is obtained by introducing redundancy in the code and in the data, without modifying the underlying hardware. This chapter provides an overview of the methods resorting to this technique, outlining their characteristics and summarizing their advantages and limitations.

9.1 Introduction

The adoption of processor-based systems even in safety- and mission-critical applications, where misbehaviors can endanger the users or cause significant economical losses, is raising new challenges for designers, who are asked to face at the same time the safety constraints with the stringent cost requirements that are typical of many application domains.

Several techniques are known since decades to meet safety constraints, which are based on a massive use of redundant (hardware and/or software) modules, and which have been developed to cope with the stringent dependability requirements of developers of traditional mission- or safety-critical applications, such as those belonging to the nuclear, military, and space domains.

However, these techniques can hardly be exploited today (at least in their original version) due to a number of constraints coming either from the new safety-critical

M. Sonza Reorda (✉), M. Rebaudengo, and M. Violante
Politecnico di Torino, Dip. di Automatica e Informatica, Corso Duca degli Abruzzi 24, 10129 Torino, Italy
e-mail: matteo.sonzareorda@polito.it; maurizio.rebaudengo@polito.it; massimo.violante@polito.it

M. Nicolaidis (ed.), *Soft Errors in Modern Electronic Systems*,
Frontiers in Electronic Testing 41, DOI 10.1007/978-1-4419-6993-4_9,
© Springer Science+Business Media, LLC 2011

application domains where processor-based systems are used, or from new technological or commercial issues that designers cannot neglect.

On one hand, the budget available to developers of mission- or safety-critical applications (especially in new market domains such as automotive, biomedical, or telecom) is shrinking constantly, to the point that commercial-off-the-shelf components, which are not specifically designed, manufactured, and validated for being deployed in critical applications, are nowadays mandatory to cut costs.

On the other hand, the advancements in manufacturing technologies have made available deep-sub-micron circuits that pack tens of millions of transistors, operate in the GHz domain, and are powered by low voltage supplies: these advancements open the frontiers for unprecedented low-cost computing power, but whose noise margins are so reduced that the obtained products (if not suitably modified) are expected to experience serious reliability problems when deployed in the field, even if they are designed, manufactured, and operated correctly. In fact, for such a kind of systems, the correct behavior has to be guaranteed taking into account the high chance that soft errors may occur while they are operating in the field. For such a kind of commodity applications, cost is one of the primary concerns, and therefore the techniques for guaranteeing high dependability coming from traditional critical domains are not affordable.

Finally, developers of mission- or safety-critical applications can benefit from the usage of commercial-off-the-shelf components not only for reasons related to component's cost, but also for performance reason. Indeed, commercial-off-the-shelf components are often one generation behind their hardened counterparts (i.e., components that are certified for being deployed in critical applications), which means that they are in general more powerful, less power demanding, etc. As a result, developers of mission- or safety-critical applications can produce better designs by using commercial-off-the-shelf components, provided that they have a low-cost way to design dependable systems from unreliable components.

The quest for reducing development costs, while meeting high dependability requirements, has seen the rise of a new design paradigm known as *software-implemented hardware fault tolerance* (SIHFT) for developing processor-based systems the user can reasonably rely on. According to this paradigm, commercial-off-the-shelf processors are used in combination with specially crafted software. This software is suitably developed, so that it can accomplish two tasks. From one side, it performs the original functionalities the designers implemented to satisfy the user's requirements, and from the other, it supports special functionalities that detect, signal, and possibly correct the occurrence of both permanent and transient hardware errors.

By using commercial-off-the-shelf processors, the designers can use the state-of-the-art components that guarantee the best performance available at the moment. Moreover, designers can cut costs significantly: commercial-off-the-shelf components come indeed at much lower costs than their hardened counterparts. Moreover, hardware redundancy is not used, since all the tasks needed to provide dependability are demanded to the software.

According to the SIHFT paradigm, the software becomes the most critical part of the system, since it is its duty to supervise the correct behavior of the whole system,

and possibly intervene to mitigate the effects of errors. On the one hand, the software must be correct, i.e., it should implement the specifications correctly. Moreover, it should be effective in coping with errors affecting the underlying hardware.

Software-implemented hardware fault tolerance may represent a viable solution to the problem of developing processor-based systems that balance costs with dependability requirements. Since many different approaches are available, designers willing to adopt them may have difficulties in selecting the approach (or the approaches) that best fits with the design's requirements.

This chapter aims at providing an overview of the most important available techniques, showing their advantages and underlining their disadvantages. We organized it as follows: Sect. 9.2 presents the approaches that are available for hardening the data that a processor-based system elaborates. This section deals with all those errors that modify the results a program computes, but that do not modify the sequence in which instructions are executed: several techniques are presented to guarantee that the system is able to detect the occurrence of these errors. Sect. 9.3 concentrates on the many approaches dealing with the problems of identifying the errors that may affect the execution flow of a program, thus changing the sequence in which the instructions are executed. Sect. 9.4 illustrates the approaches that allow developing fault-tolerant systems, where errors are both detected and corrected. Finally, Sect. 9.5 draws some conclusions.

9.2 Addressing Faults Affecting the Data

When considering the possible effects of faults that can arise in a processor-based system, a common categorization distinguishes between faults that simply change the data the application works on (thus forcing the code to produce wrong results, while following the same execution flow), and faults that affect the control flow, e.g., by changing the op-code of an instruction just before it is decoded and executed. This section focuses on the methods for facing faults belonging to the former category.

The adoption of these techniques is generally rather expensive in terms of memory size, execution slow down, and programming limitations. On the other side, they generally offer a very high coverage of the addressed faults.

The methods considered in this section mainly belong to two categories:

1. Those based on computation duplication;
2. Those based on assertions.

9.2.1 Computation Duplication

The basic idea behind this group of techniques is that the operations performed by the code can be duplicated, and the produced results compared, thus allowing the

detection of possible faults. Duplication may be implemented at four levels of granularity: instruction, instructions block, procedure, or program.

Starting from the smallest granularity, duplication can be adopted at the instruction-level, where an individual instruction is duplicated. For example, the duplicated instruction is executed immediately after the original instruction; the duplicated instruction may perform the same computation carried out by the original instruction, or it can perform a modified version of the original operation.

The coarsest level of duplication is the program level, in which the whole program is duplicated: the duplicated program may be executed after the original program completes its execution, or it can be executed concurrently.

Whatever the level of granularity adopted, the technique is able to detect faults by executing a check after the duplicated code is executed. With the instruction-level duplication, a check compares the results coming from the original instruction and its duplication; with the procedure-level duplication, the results of the duplicated procedures are compared; with the program-level duplication, a comparison among the outputs of the programs is executed in order to detect possible faults.

In order to give the reader an idea of this group of methods, we will first focus on *instruction-level duplication*. A simple method belonging to this category, and able to achieve a satisfactory error detection capability, has been originally proposed in [1]; it is based on introducing data and code redundancy according to a set of transformations to be performed on the high-level source code. The transformed code is able to detect errors affecting both data and code: the goal is achieved by duplicating each variable and adding consistency checks after every read operation. Other transformations focus on errors affecting the code, and correspond from one side to duplicating the code implementing each write operation, and from the other, to adding checks for verifying the consistency of the executed operations.

The main advantage of the method lies in the fact that it can be automatically applied to a high-level source code [2], thus freeing the programmer from the burden of guaranteeing its correctness and effectiveness (e.g., by selecting what to duplicate and where to insert the checks). The method is completely independent on the underlying hardware, and it possibly complements other already existing error detection mechanisms.

The rules mainly concern the variables defined and used by the program. The method refers to high-level code only and does not care whether the variables are stored in the main memory, in a cache, or in a processor register. The proposed rules may complement other Error Detection Mechanisms that can possibly exist in the system (e.g., based on parity bits or on error correction codes stored in memory). It is important to note that the detection capabilities of the rules are significantly high, since they address any error affecting the data, without any limitation on the number of modified bits or on the physical location of the bits themselves.

The basic rules can be formulated as follows:

- *Rule 1*. Every variable x must be duplicated: let x_0 and x_1 be the names of the two copies.
- *Rule 2*. Every write operation performed on x must be performed on x_0 and x_1.

- *Rule 3*. After each read operation on x, the two copies x_0 and x_1 must be checked for consistency, and an error detection procedure should be activated if an inconsistency is detected.

The check must be performed immediately after the read operation, in order to block the fault effect propagation. Please note that variables should also be checked when they appear in any *expression* used as a condition for branches or loops, thus allowing a detection of errors that corrupt the correct execution flow of the program.

Every fault that occurs in any variable during the program execution can be detected as soon as the variable is the source operand of an instruction, i.e., when the variable is read, thus resulting in minimum error latency, which is approximately equal to the temporal distance between the fault occurrence and the first read operation. Errors affecting variables after their last usage are not detected (but do not provoke any failure, too).

Two simple examples are reported in Fig. 9.1, which shows the code modifications for an *assignment* operation and for a *sum* operation involving three variables a, b, and c.

Experimental results reported in [3] show that the method is able to provide a nearly complete fault detection capability; on the other side, it introduces a significant memory overhead (about 3 times with respect to the original program) and a comparable slow-down factor.

In [4], a method has been introduced that is able to select a subset of variables to be duplicated; in this way, the cost of the method can be significantly improved, at the expense of a small reduction in the fault detection capabilities.

An interesting method based on similar concepts and named EDDI (*Error-Detection by Duplicated Instructions*) has been presented in [5]. EDDI aims at exploiting the architecture of modern superscalar processors to achieve fault detection capabilities. In particular, the authors of EDDI propose to duplicate the original instructions in the original assembly source code using duplicated registers and variables, too. When executed by a superscalar architecture, it is likely that the cost for executing the hardened code is less than doubled with respect to the original one, since they eventually exploit the idle functional units within the pipeline.

Original code	*Modified Code*

`a = b;`	`a0 = b0;`
	`a1 = b1;`
	`if (b0 != b1)`
	`  error ();`

`a = b + c;`	`a0 = b0 + c0;`
	`a1 = b1 + c1;`
	`if ((b0!=b1) || (c0!=c1))`
	`  error ();`

Fig. 9.1 Example of code modification

Following a similar idea, detection capability can also be obtained exploiting idle cycles of a fine-grain parallel architecture composed of multiple pipelined functional units, where each functional unit is capable of accepting an instruction in every clock cycle. This approach is called *instruction re-execution* [6], and it addresses the performance degradation caused by time redundancy. With instruction re-execution, the program is not explicitly duplicated. Rather, when an instruction reaches the execution stage of the processor pipeline, two copies of the instruction are formed and issued to the execution units. Since instructions are duplicated within the processor itself, the processor has flexible control over the scheduling of redundant computations. Dynamic scheduling logic combined with a highly parallel execution core allows the processor to exploit idle execution cycles and execution units to perform the redundant computations.

Devising instruction duplication techniques for Very Long Instruction Word (VLIW) processors has been explored by Bolchini [7]. The experimental results showed the performance degradation ranges from 2 to 25% with respect to the nominal application code. The limited impact can be related to the low average number of operations per clock cycle, which leaves several empty slots for duplicated operations.

9.2.2 Procedure-Level Duplication

9.2.2.1 Selective Procedure Call

The *Selective Procedure Call Duplication* (SPCD) [8] technique is based on the duplication of the procedure execution. Every procedure (or a selected subset of them) is called twice with the same parameters; the results produced by the two executions are stored and compared, thus allowing for error detection. Figure 9.2 shows an example where procedure B is called twice.

The main goal of this approach is the improvement of the systems' reliability by detecting transient errors in hardware, taking into account the reduction of the energy consumption and of the overhead.

SPCD minimizes energy dissipation by reducing the number of clock cycles, cache accesses, and memory accesses by selectively duplicating procedure calls instead of duplicating every instruction. The number of additional clock cycles is reduced because the number of comparisons is reduced by checking the computation results only after the original and duplicated procedure execution (instead of checking the results immediately after executing every duplicated instruction). The code size is also reduced because the procedure code is not duplicated. The probability of an instruction cache miss can also be lowered in this way (since the data and code of a procedure may remain in the caches from one call to the following) and energy consumption can be reduced for fetching instructions from the cache to the processor, or moving instructions from the memory to the cache. Moreover, reducing the number of comparisons decreases the number of

```
int a, a1, b, c, c1;
void A2 ()
{
    a = B(b);
    a1 = B(b);
    if (a <> a1) errorHandler();
    c = c + a;
    c1 = c1 + a1;
    if (c <> c1) errorHandler();
}
int B(int b)
{
int d;
    d = 2 * b;
    return(d);
}
```

Fig. 9.2 Procedure-level duplication

data accesses to the data cache and the memory, resulting in reduced energy consumption.

However, there is a trade-off between energy saving and error detection latency: longer error detection latency reduces the number of comparisons inside the procedure, and therefore, saves energy. The shortest error detection latency can be achieved by instruction-level duplication. In procedure-level duplication, comparison of the results is postponed until after executing the called procedure twice; then, the worst case error detection latency corresponds to the execution time of the original and duplicated procedures and the comparison time.

The adoption of the SPCD technique may require some additional attention to deal with special cases. As an example, if a procedure modifies a global variable, duplicating its execution can introduce an incorrect behavior. This may require changing the procedure code to avoid the problem.

SPCD was simulated with some benchmark programs.

Fault injection experiments were executed injecting single-bit flip faults in the adder unit. Experimental results show the following:

- As the error detection latency increases, the energy consumption is reduced.
- The data integrity (i.e., the correctness of the outputs) reported is always 100%.
- The number of detected faults decreases as the error detection latency increases, but the undetected faults do not cause any failure because they do not affect the final results.

In order to evaluate the feasibility of the approach in terms of energy consumption saving, SPCD is compared with the hardened program obtained by applying an instruction-level duplication approach [5]. The obtained results show that SPCD allows an energy saving of 25% with respect to the energy consumption required by an instruction-level duplication approach.

9.2.3 Program-Level Duplication

9.2.3.1 Time Redundancy

When time redundancy is adopted, the same computation is performed multiple times (possibly in different forms) on the same hardware. A particular application of time redundancy is the *duplication* of the processing activity as a proper technique to detect faults of the underlying hardware. A particular form of such duplication is the Virtual Duplex System (VDS) architecture, where the duplicity is achieved by temporal redundancy, which is obtained by executing two programs performing the same task with the same input data. Virtual duplex systems provide a cost advantage over duplex systems because of reduced hardware requirements. VDS only needs a single processor, which executes the same program twice, or two programs in sequence implementing the same algorithm. In the former case, transient hardware errors are covered due to time redundancy, as only a single execution is affected. In the latter case also, permanent hardware errors can be detected due to design diversity. For these reasons, the programs of a VDS are diversified in order to reduce the probability that both of them are affected in the same way.

The disadvantage of time redundancy is the performance degradation caused by repetition of tasks.

There are different kinds of duplication: one option consists in running entire programs twice, whereby another option is to execute the duplicated processes in short rounds and switch between them. The switching introduces extra overhead, but can be used to compare intermediate results more frequently in order to reduce the fault latency.

The structure of a VDS is reported in Fig. 9.3. Each version of a program is called *variant*. A VDS built to calculate a specified function f consists of two diversified program variants P_a and P_b calculating the functions f_a and f_b, respectively, on the same input values i. In absence of faults, $f = f_a = f_b$ holds. If a fault affects only one of the two variants or both of them in different ways, then the fault can be detected comparing the results $f_a(i)$ and $f_b(i)$.

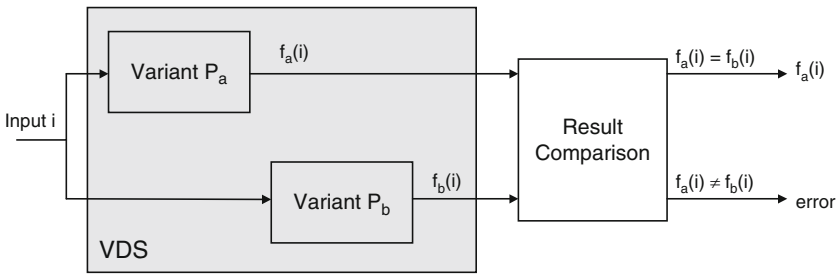


Fig. 9.3 Structure of a VDS

The kind of faults detected by a VDS highly depends on the diversity techniques used to generate the VDS.

For example, if two independent teams develop different variants of a program, then the resulting VDS may have the ability to detect both specification and implementation faults. If, as a second example, two different compilers are used to compile the same source code, then the resulting VDS may have the ability to detect faults stemming from compiler faults. The capability of diversified program variants to detect hardware faults has been mentioned and investigated in [9]. The basic idea is that two diversified programs often use different parts of the processor hardware in different ways with different data.

Variants can also be generated by manually applying different diversity techniques. However, some algorithmic approaches have been proposed in order to properly generate effective software variants.

In [10], a systematic method is presented based on the transformation of every instruction of a given program into a modified instruction or sequence of instructions, keeping the algorithm fixed. The transformations are based on a diverse data representation. Since a diverse data representation also requires a modification of the instructions that have to be executed, new sequences of instruction must be generated, that calculate the result of the original instruction in the modified representation. The transformations can be generated at the assembly level as well as in at a higher level. Some examples of the modification rules are as follows:

- Logical instructions can be modified according to the de Morgan Rules (e.g., a or b = NOT (NOT (A) AND NOT (B))).
- Arithmetic instructions can be modified according to the two's complement properties (a + b = − (−a) + (−b)).

In [11], a method for the automated generation of variants is proposed. The tool is able to generate two different, but semantically equivalent, pieces of assembly code, exploiting a set of modification rules. Some examples of modification rules are as follows:

- Replacement of jump instructions (e.g., replacement of conditional jump instructions by appropriate combinations of other jump instructions);
- A consistent register permutation;
- Substitution of a multiplication statement by a subroutine that performs multiplication in a different way.

9.2.3.2 Simultaneous Multithreading

Simultaneous multithreading (SMT) is a novel technique to improve the performance of a superscalar microprocessor. A SMT machine allows multiple independent threads to execute simultaneously, i.e., in the same cycle, in different functional units. VDS can be effectively exploited on a SMT machine, executing two threads in

parallel, shifting time redundancy to spatial redundancy [12]. Because of the improved processor utilization and the absence of a context switch, the time execution is reduced with respect to the correspondent duplicated implementation on a conventional processor.

With the *Active-Stream/Redundant-Stream Simultaneous multithreading* (AR-SMT) [13] approach, two explicit copies of the program run concurrently on the same processor resources as completely independent programs, each having its own state or program context. The entire pipeline of the processor is conceptually duplicated. As already mentioned before, in superscalar processors often there are phases of a single program that do not fully utilize the microprocessor architecture, so sharing the processor resources among multiple programs will increase the overall utilization. Improved utilization reduces the total time required to execute all program threads, despite possibly slowing down single thread performance. AR-SMT is based on two streams: active stream (A-stream) and redundant instruction stream (R-stream). The active stream corresponds to the original program thread and as instructions from the A-stream are fetched and executed, and their results committed to the program's state, the results of each instruction are also pushed on a FIFO queue called *Delay Buffer*. Results include modifications to the Program Counter by branches and any modifications to both registers and memory. The second stream (R-stream) is executed simultaneously with the A-stream. As the R-stream is fetched and executed, its committed results are compared to those stored in the Delay Buffer. A fault is detected if the comparison fails, and the committed state of the R-stream can be used as a checkpoint for recovery. Simulations made on a processor composed of 8 Processing Elements show that AR-SMT increases execution time by only 10–40% over a single thread, thanks to the optimized utilization of the highly parallel microprocessor.

9.2.3.3 Data Diversity

The method exploits data diversity by executing two different programs with the same functionality, but with different data sets and comparing their outputs. This technique is able to detect both permanent and transient faults.

This approach produces two different programs starting from the original one; a new version is generated by multiplying all variables and constants by a *diversity factor k*. Depending on the factor k, the original and the transformed program may use different parts of the underlying hardware, and propagate fault effects in different ways. If the two programs produce different outputs due to a fault, the fault can be detected by examining if the results of the transformed program are also k times greater than the results of the original program. The check between the two programs can be executed in two different ways:

1. Another concurrent running program compares the results.
2. The main program that spawns the original program and the transformed program checks their results after they are completed.

In Fig. 9.4, a sample program is transformed into a diverse program, where $k = -2$.

The choice of the most suitable value for k is performed by trying to maximize the probability that the two programs produce different outputs for the same hardware fault. However, the factor k should also not cause an overflow in the functional units. The overflow problem can be solved by scaling: scaling up the data of the modified program to higher precision, or scaling down the original data. Scaling up may cause performance overhead because the size of the data is doubled. On the other hand, scaling down the original data (the same effect as dividing the original data by k instead of multiplying by k) will not cause any performance overhead. However, there is a possibility that scaling down data may cause computation inaccuracy during the execution of the program. In this case, when the scaled down values are compared with the original values, only the higher order bits that are not affected by scaling down have to be compared.

A first method [10] proposed to consider $k = -1$, i.e., data are negated.

The method proposed in [14], called ED^{4}I (*Error Detection by Diverse Data and Duplicated Instructions*), demonstrated that in different functional units, different values of k maximize the fault detection probability and data integrity (e.g., the bus has the highest fault detection probability when $k = -1$, but the array multiplier has the highest fault detection probability when $k = -4$). Therefore, programs that use a particular functional unit extensively need preferably a certain *diversity factor* k. Considering six benchmark programs (Hanoi, Shuffle, Fibonacci, Lzw compression, Quick sort, and Insert sort), the most frequently used functional units are adders and $k = -2$ is the optimum value. On the other hand, the matrix multiplication program extensively uses the multiplier and the optimum value is $k = -4$.

This hardening technique introduces a memory overhead higher than two times the memory required for the original program and the performance overhead is also higher than 2.

ED^{4}I is applicable only to programs containing assignments, arithmetic operations, procedure calls, and control flow structures, and cannot be applied to statements executing logic operations (e.g., Boolean functions, shift or rotate operations) or exponential or logarithmic functions.

Original version	Transformed version

`x = 1;`
`y = 5;`
`i = 0;`
`while (i < 5){`
`  z = x + i * y;`
`  i = i + 1;`
`  }`
`i = 2 * z;`
	`x = -2;`
	`y = -10;`
	`i = 0;`
	`while (i > -10) {`
	`  z = x + i * y /(-2);`
	`  i = i + (-2);`
	`}`
	`i = (-4) * z / (-2);`

Fig. 9.4 Sample program in the original and transformed version

9.2.4 Executable Assertions

This method is based on the execution of additional statements that check the validity and correctness of the data corresponding to the program variables.

The effectiveness of executable assertions method is highly application dependent. Moreover, in order to identify the most suitable executable assertions, the developers require extensive knowledge of the system.

Error detection in the form of executable assertions can potentially detect any error in internal data caused by software faults or hardware faults.

The approach proposed in [15] describes a rigorous way of classifying the data to be tested. The two main categories in the classification scheme are *continuous* and *discrete* signals. These categories have subcategories that further classify the signal (e.g., the continuous signals can be divided into monotonic and random signals). For every signal class, a specific set of constraints is set up, such as boundary values (maximum and minimum values) and rate limitations (minimum and maximum increase or decrease rate), which are then used in the executable assertions. Error detection is performed as a test of the constraints. A violation of a constraint is interpreted as the detection of an error.

Executable Assertion and best effort recovery are proposed in [16], considering a control application. The state variables and outputs are protected by executable assertions to detect errors using the physical constraints of the controlled object. The following erroneous cases can be detected:

- If an incorrect state of the input variable is detected by an executable assertion during one iteration of the control algorithm, a recovery is made by using the state backed-up, during the previous iteration of the computation. This is not a true recovery, since the input variable may differ from the value used in the previous iteration. This may result in the output being slightly different from the fault-free output, thus creating a minor value failure (*best effort recovery*).
- If an incorrect output is detected by an executable assertion, recovery is made by delivering the output produced in the previous iteration. The state variable is also set to the state of the previous iteration that corresponds to the delivered output. This is a best effort recovery too, since the output could be slightly different from the fault-free value.

The approach based on executable assertions with best effort recovery has been experimentally applied on an embedded engine controller [16]. Fault injection experiments executed on the original program showed that 10.7% of the bit-flips injected into data cache and internal register of a CPU caused a failure in the system. Fault injection experiments run on the hardened program modified with the executable assertions with best effort recovery showed that the percentage of failures decreased to 3.2%, demonstrating that software assertions with best effort recovery can be effective in reducing the number of critical failures for control algorithms.

9.3 Addressing Faults Affecting the Execution Flow

This section presents the main software-implemented techniques for hardening a processor-based system against control flow (CF) errors (CFEs). A CFE is an error that causes a processor to fetch and execute an instruction different than expected.

First of all, we will introduce some general concepts and definitions. Afterwards, the most representative methods among those proposed to face CFEs will be overviewed.

9.3.1 Background

The program code can be partitioned into basic blocks (BBs). A *BB* (sometimes also named *branch free interval*) of a program is a maximal sequence of consecutive program instructions that, in absence of faults, are always executed altogether from the first one up to the last one.

So, a BB does not contain any instruction that may change the sequential execution, such as jump or call instructions, except for the last one, possibly. Furthermore, no instructions within the BB can be the destination of a branch, jump or call instruction, except for the first one, possibly.

The *BB body* is the BB without the last jump instruction. If the last BB instruction is not a jump instruction, then the BB body coincides with the BB. It may happen that a BB body is empty if the BB consists of one jump instruction only.

A program P can be represented with a CF Graph (CFG) composed of a set of nodes V and a set of edges B, $P = \{V, B\}$, where $V = \{v_1, v_2, \ldots, v_n\}$ and $B = \{b_{i1,j1}, b_{i2,j2}, \ldots, b_{im,jm}\}$. Each node $v_i \in V$ represents a program section, which can correspond to a single instruction or a BB. Each edge $b_{i,j} \in B$ represents the branch from node v_i to node v_j [17].

As an example, let us consider the sample program fragment shown in Fig. 9.5 (where the BBs are numbered, and the corresponding program CFG is shown).

Considering the program CFG $P = \{V, B\}$, for each node v_i, it is possible to define suc(v_i) as the set of nodes successor of v_i and pred(v_i) as the set of nodes predecessor of v_i [18]. A node v_j belongs to suc(v_i) if and only if $b_{i,j}$ is included in B.

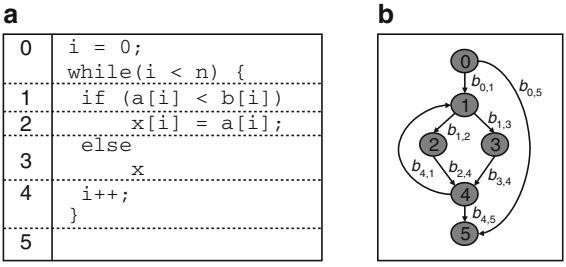


Fig. 9.5 Example of program source code and its CFG

Similarly, v_j belongs to pred(v_i) if and only if $b_{j,i}$ is included in B. For example, in the CFG from Fig. 9.5b suc$(1) = \{2, 3\}$ and pred$(1) = \{0, 4\}$.

Let a program be represented by its CFG $P = \{V, B\}$. A branch $b_{i,j}$ is *illegal* for P if $b_{i,j}$ is not included in B [17]. If a program, due to a fault, executes a branch $b_{i,k} \in B$ instead of the correct branch $b_{i,j}$, then the branch $b_{i,k}$ is *wrong*. Illegal and wrong branches correspond to CFEs.

CFEs can be divided into *intra-block* errors, which cause erroneous branches having as their source and destination different blocks, and *inter-block* errors, which cause erroneous branches not crossing the blocks boundaries. Correspondingly, intra-block Control Flow Check (CFC) techniques control that instructions inside a block are executed in the correct order, and inter-block CFC techniques detect inter-block CFEs.

A common approach for the software-implemented detection of CFEs, causing erroneous branches inside program area, is the signature-monitoring technique. In this approach, monitoring is performed by regular processor instructions (called *monitoring* or *checking code*) embedded into the program under execution. A signature (or identifier) is associated to program structure (it can be a singular instruction, a block of instructions, a path of the program CFG, or other) during the compile time or by special program prior to program execution. During the program execution, a run-time signature is computed. Periodically, checks for consistency between the reference signature and the run-time signature are performed. The mismatch signals the CFE. The run time signature computed during the program execution is usually stored in a special area (e.g., a processor register).

In this section, we considered the fault models proposed in literature (e.g., [18–20]) including the following types of CFEs:

- *Type 1*: a fault causing an illegal branch from the end of a BB to the beginning of another BB;
- *Type 2*: a fault causing a legal but wrong branch from the end of a BB to the beginning of another BB;
- *Type 3*: a fault causing a branch from the end of a BB to any point of another BB body;
- *Type 4*: a fault causing a branch from any point of a BB body to any point of different BB body;
- *Type 5*: a fault causing a branch from any point of a BB body to any point in the same BB body.

The first four types represent inter-block CFEs, while type 5 represents intra-block CFEs. The considered fault model includes only CFEs, which lead to erroneous branches inside the program memory.

9.3.2 ECCA

In order to reduce the overhead, the method known as *Enhanced Control flow Checking using Assertions* (ECCA) [19] divides the program into a set of blocks,

where the block is a collection of consecutive BBs (BBs are called *Branch Free Interval* or BFI in [19], but we will hold on the accepted terminology) with single entry and single exit. The shorter the block is, the higher the fault coverage is and the lower the error detection latency is, whereas the memory and performance overhead is higher. By properly choosing the block length, it is possible to find the most suitable trade-off for the user purposes.

Error detection in ECCA is performed resorting to the exception handler. ECCA assigns a unique prime number identifier (called *Block Identifier* or BID) greater than 2 to each block of a program. During program execution, the global integer variable id is updated to contain the currently traversed block identifier.

Two assertions are added to each block:

1. A SET assertion is added at the beginning of the block, which executes two tasks: it assigns the BID of the current block to the id variable and it checks if the block from which the execution came from is a predecessor block, according to the CFG. A divide by zero error signals a CFE. The SET assertion implements the following formula:

$$id = \frac{BID}{\overline{(id \bmod BID)} \cdot (id \bmod 2)}, \tag{9.1}$$

where,

$$\overline{(id \bmod BID)} = \begin{cases} 1, & if\,(id \bmod BID) = 0, \\ 0, & if\,(id \bmod BID) \neq 0. \end{cases}$$

2. A TEST assignment is executed at the end of the block and executes two tasks: it updates the id variable taking into account the whole set of successor according to CFG blocks and checks if the current value of the id variable is equal to BID. The TEST assertion implements the following formula:

$$id = NEXT + \overline{\overline{(id - BID)}}. \tag{9.2}$$

The variable NEXT is equal to the product of BIDs of all successors according to the CFG blocks of the current block, i.e.,

$$NEXT = \prod BID_{successor}, \tag{9.3}$$

$$\overline{\overline{(id - BID)}} = \begin{cases} 1, & if\,(id - BID) \neq 0, \\ 0, & if\,(id - BID) = 0. \end{cases}$$

The *NEXT* and *BID* variables are generated once before the program execution, whereas the id variable is updated during the program execution.

Experimental analysis demonstrates that the main merit of the approach is its high CFE coverage. ECCA covers all single CFEs of types 1, 3, and 4. Legal but

wrong branches (errors of type 2) as well as intra-block CFEs (errors of type 5) are not considered by the method.

The drawback of the method is the quite high memory and performance overhead: although only two instructions for block are added in ECCA, these instructions are rather complex and are translated in a high number of instructions in the executable assembly code.

9.3.3 CFCSS

In [18], an assembly-level CFC approach named *Control Flow Checking by Software Signatures* (CFCSS) is proposed.

CFCSS assigns a unique arbitrary number (signature) s_i to each BB. During program execution, a run-time signature G is computed in each BB and compared with the assigned signature. In the case of discrepancy, a CFE is detected. A run-time signature G is stored in one of the general-purpose registers (GSR).

At the beginning of the program, G is initialized with the signature of the first block of the program. When a branch is taken the signature G is updated in the destination BB v_i using the signature function f. The signature function f is computed resorting to the following formula:

$$f(G, d_i) = G \oplus d_i, \tag{9.4}$$

where,

$$d_i = s_j \oplus s_i, \tag{9.5}$$

and s_j is the signature of the predecessor of the BB v_i. d_i is calculated in advance during the compilation and stored in the BB v_i.

For example, if the currently traversed BB is s_j, then $G = s_j$; when the control is passed to the BB s_i, G is updated as follows:

$$G = G \oplus d_i.$$

Substituting the values of G and d_i, we have

$$G = s_j \oplus s_j \oplus s_i = s_i.$$

Therefore, in the absence of CFEs the variable G contains the signature of the currently traversed BB.

If a CFE happened, leading to an illegal branch from BB v_k to v_i (whose predecessor is the BB v_j), then $G = G \oplus d_i = s_k \oplus s_j \oplus s_i \neq s_i$.

At the top of each BB v_i (before the original instructions of the BB) some new code is added, which updates the signature G using the signature function f and compares the computed run-time signature with the assigned one (Fig. 9.6). In the

Fig. 9.6 Checking code

```
G = G ⊕ d_i;
if(G != s_i) error();
```

Fig. 9.7 Checking code for predecessor BB of the branch-fan-in BB

```
G = G ⊕ d_k;
if (G != s_k) error();
D = s_j ⊕ s_k;
```

Fig. 9.8 Checking code for the branch-fan-in BB. Signatures are embedded in the program during compilation or preprocessing

```
G = G ⊕ d_i;
G = G ⊕ D;
if (G != s_i) error();
```

case of mismatch, the error is detected and the control is transferred to an error handling routing.

If the BB v_i has more than one predecessor (i.e., v_i is a branch-fan-in BB), an adjusting signature D is defined in each predecessor BB of v_i and used in the BB v_i to compute the signature. The adjusting signature D is set to 0 for one arbitrary chosen predecessor BB of v_i (let it be v_j); for each BB v_k, $k \neq j$ the predecessor of v_i, the adjusting signature D is defined as $D = s_j \oplus s_k$. For the BB v_k (predecessor of the branch-fan-in BB), the checking code is presented in Fig. 9.7. For the branch-fan-in BB v_i, the checking code is presented in Fig. 9.8.

Once the CFE corrupted the CF, causing the discrepancy between run-time signature and the expected one in some program BBs, the run-time signature remains different than the expected signature also in subsequent BBs. Basing on this property, the authors of the CFCSS technique propose to perform consistency checks only in some of the program BBs, which allows to reduce the technique overhead. Postponing the check is possible only in case the error detection latency is acceptable for the application.

Experimental analysis demonstrates that the approach is capable of detecting most of CFEs of types 1, 3, and 4. It does not detect CFEs of types 2 and 5 and those CFEs of type 4, which lead from inside of some BB to the beginning of some its successor BB.

As it is shown in [18], aliasing errors are also possible for CFEs of type 1 in the case where multiple BBs share multiple BBs as their destination nodes. For example, given a program CFG with the set of edges B containing the subset $\{b_{1,4}, b_{1,5}, b_{2,5}, b_{2,6}, b_{3,5}, b_{3,6}\}$, an erroneous illegal branch $b_{1,6}$ is not detectable by the method.

9.3.4 YACCA

The *Yet Another Control flow Checking Approach* (YACCA) [20] assigns to each program BB v_i two unique identifiers $I1_i$ and $I2_i$. The identifier $I1_i$ is associated to the BB v_i entry and the identifier $I2_i$ is assigned to the BB v_i exit.

An integer variable code is introduced in the program, which stores a run-time CF signature during the program execution. The code variable is updated by means of the *set* assertion to the value of the entry identifier $I1_i$ at the beginning of the BB v_i, and to the value of the exit identifier $I2_i$ at the end of the BB v_i.

Before each set assertion, a *test* assertion is performed. At the beginning of the BB v_i, a test assertion verifies if the run-time value of the code variable corresponds to the identifier $I2_j$ of some BB belonging to the pred(v_i) set, while at the end of the BB v_i, a test assertion verifies if the run-time value of the code variable corresponds to $I1_i$.

The update of the variable code value is performed according to the following formula:

$$code = (code \ \& \ M1) \oplus M2, \tag{9.6}$$

where M_1 represents a constant mask whose value depends on the set of possible predecessor values of the code variable, whereas M_2 represents a constant mask depending both on the identifier which should be assigned to the code variable and on the possible predecessor values of the code variable. For example, the values M_1 and M_2 can be defined as follows:

- For the *set* assertion at the beginning of the generic BB v_i:

$$M_1 = \overline{\left(\underset{j:v_j \in pred(v_i)}{\&} I2_j \right) \oplus \left(\underset{j:v_j \in pred(v_i)}{\vee} I2_j \right)}, \tag{9.7}$$

$$M_2 = (I2_j \ \& \ M_1) \oplus I1_i. \tag{9.8}$$

- For the set assertion at the end of the generic BB v_i:

$$M_1 = 1,$$

$$M_2 = I1_i \oplus I2_i. \tag{9.9}$$

The binary representation of M_1 obtained by (9.7) contains the value 1 in the bits having the same values in all identifiers $I2_j$ of BBs from pred(v_i), and the value 0 in the bits having different values in these identifiers. The operation (code & M_1) allows to set the code variable to the same value I from any possible predecessor value of the code variable. Therefore, performing the XOR operation of I and M_2 allows to obtain the value $I1_i$.

To avoid the aliasing effect, the identifiers of the BBs should be chosen in such a way that the new value of the code variable is equal to the targeted value if and only if the old value of the code variable is possible according to the program CFG, i.e., the operation ($I2_j$ & M_1) should not return the value I if BB v_j does not belong to pred(v_i).

The test assertion introduced at the beginning of the BB v_i with pred $(v_i) = \{v_{j1}, v_{j2}, \ldots, v_{jn}\}$ is implemented as follows:

$$\text{ERR\_CODE}| = \big((\text{code}! = \text{I2}_{\text{j1}}) \;\&\&\; (\text{code}! = \text{I2}_{\text{j2}}) \;\&\&\; (\cdots) \;\&\&\; (\text{code}! = \text{I2}_{jn})\big), \tag{9.10}$$

where the ERR_CODE variable is a special program variable containing the value 1 if the CFE is detected and 0 otherwise. The ERR_CODE variable is initialized with the value 0 at the very beginning of the program execution.

The *test* assertion introduced at the end of the BB v_i is implemented as follows:

$$\text{ERR\_CODE}| = (\text{code}! = \text{I1}_{\text{i}}). \tag{9.11}$$

In order to identify the wrong branches, the test is repeated for each conditional branch at the beginning of both the true and the false clause. In order to identify all wrong branches, each condition should contain the "else" clause; if the "else" clause is absent, it should be introduced and the corresponding BB should contain *test* and *set* assertions.

An experimental analysis demonstrated the effectiveness of the YACCA method as far as the fault coverage is considered. A very limited number of CFEs cause a failure, and the method shows itself to be more powerful than the considered alternative approaches. The method covers all single errors of the types 1–4. Elaboration time is limited since the set and test assertions do not involve divisions or multiplications.

9.3.5 CEDA

The *Control-flow Error Detection through Assertions* (CEDA) [21] approach considers two different node types (NT):

1. $NT = A$: if it has multiple predecessors and at least one of its predecessors has multiple successors.
2. $NT = X$: otherwise.

A register contains the value of a signature S, which is continuously updated to monitor the execution of the program.

For each node, two different signatures are statically assigned:

1. *NS*: the expected value of S at any point within the node on correct execution of the program
2. *NES*: the expected value of S on exiting the node on correct execution of the program

Two parameters (d_1 and d_2) are associated statically with each node, and are used to update S in order to match the expected values NS and NES.

At run-time, S is updated at the entry and at the exit of each node.

At the beginning of execution of a node v_i, S is updated as follows:

- $S = S \text{ AND } d_1(v_i)$, if $NT(v_i) = A$.
- $S = S \text{ XOR } d_1(v_i)$, if $NT(v_i) = X$.

According to this update, the value of S should match the expected valued assigned to the signature of the node, $NS(v_i)$.

At the end of each node v_i, S is updated as follows:

$$S = S \text{ XOR } d_2(v_i)$$

According to this update, the value of S should match the expected value assigned to the exiting signature of the node, $NES(v_i)$.

Check instructions are inserted at certain points in the program (called *check points*). A check instruction S is compared against its expected value at that point.

Experimental analysis demonstrates that the achieved fault coverage is comparable with respect to that reached by the other state-of-the-art approaches, while a lower performance overhead is introduced by the method.

In order to further reduce the memory and performance overheads for real-time embedded systems, the following techniques have been introduced in [22]:

- The frequency of signature updates can be reduced by adopting the concept of *supernode* that corresponds to a collection of nodes; the program code modifications (whichever they are) are applied to the supernodes, instead of to the nodes.
- Exploiting the principle of locality, it is possible to demonstrate that each program spends a majority of time executing a small number of functions; thus it is possible to *tightly* protect the few functions which are executed for the majority of time and *loosely* protect the rest of functions.

The application of this technique reduces the fault coverage by a negligible value (i.e., from 99.3 to 99%), but the memory and performance overheads can be reduced by as much as 50%.

9.4 Addressing Fault Tolerance

When fault tolerance (and not only fault detection) capabilities are the target, the approaches presented so far are not enough. Some of them can be extended (at a higher cost in terms of memory, performance, and development cost) to cope with the more stringent requirements, and new approaches can be devised to address these requirements. Obviously, the same assumptions holding for the previous sections are valid here: therefore, we will focus on techniques that allow reaching fault tolerance resorting only to changes in the software. The techniques that are covered in this section are design diversity, checkpointing, algorithm-based fault tolerance, and duplication.

9.4.1 Design Diversity

The concept of design diversity is very old. At the beginning of the XIX century Charles Babbage has suggested that "the most certain and effectual check upon errors which arise in the process of computation is to cause the same computation to be made by separate and independent computers; this check is rendered still more decisive if they make their computations by different methods" [23]. This theory can be transferred and adapted easily to the modern computer science.

The use of redundant copies of hardware, data and programs' instruction has proven to be quite effective in the detection of physical faults and in subsequent system recovery. However, design faults – which are introduced by human mistakes or defective design tools – are reproduced when redundant copies are made. *Design diversity* is the approach in which the hardware and software elements that are to be used for multiple computations are not copied, but are independently designed to fulfill the same function through implementations based on different technologies. A definition of design diversity has been given in [24] as "production of two or more systems aimed at delivering the same service through separate designs and realizations."

Design diversity is the common technique adopted to achieve software fault tolerance. Two or more versions of software are developed by independent teams of programmers and software engineers, and by using different techniques, methodologies, algorithms, programming languages and programming compilers. However, all the different implementations of the software meet the common requirements and specifications.

The versions of software produced through the design diversity approach are called (as already introduced in Sect. 9.2.3) *variants* (or *versions* or *alternates*). Besides the existence of at least two variants of a system, tolerance of faults needs a *decider* (or *acceptance test*), aimed at providing an error-free result from the variants execution; the variants execution have to be performed from consistent initial conditions and inputs. The common specification has to address explicitly the *decision points* defined as follows:

- The time when the decisions have to be performed
- The data processed by the decider

The two most common techniques implementing design diversity are *N-Version Programming (NVP)* [25] and *Recovery Blocks (RB)* [26]. These techniques have mainly been introduced to face the effects of software bugs. However, they can also be adopted to address hardware faults; they do not depend on any particular error model and are able to detect (and in some cases correct) both transient and permanent errors.

N-version programming requires the separate, independent preparation of multiple (i.e., *N*) versions of a program for some applications. These versions are executed in parallel. At the system level, an application environment controls their execution. Each one receives identical inputs and each produces its version of the required outputs. A voter collects the outputs that should, in principle, all be the same (*consensus*). If the outputs disagree, the system detects the error and can

tolerate it using the results of the majority, provided there is one. *N*-version programming is easily classified as a *static redundant scheme* and presents many analogies with the triple modular redundancy (TMR) and the *N*-modular redundancy (NMR) approaches used for tolerating hardware failures [27].

The decision mechanism is defined as follows:

- A set of program-state variables is to be included in a comparison vector (*c*-vector); each program stores its *c*-vector.
- The *N* programs possess all the necessary attributes for concurrent execution.
- At a specified cross-check point, each program generates its comparison vector and a decider executes the decision algorithm comparing the *c*-vectors and looks for the consensus of two or more *c*-vectors among the *N* versions.

The *N* independent versions of the software can be run in sequence (i.e., one after another) on a single computer; alternatively, they could be run simultaneously on independent computers.

This method has been widely exploited to target possible software errors, i.e., design faults or software bugs [28], but can be effectively adapted to detect possible hardware errors. A transient error in a hardware component has the same effect as a software one. If any of the hardware components experiences an error, it causes the software version running on that hardware to produce inaccurate results. If all of the other hardware and software modules are functioning properly, the system will still produce a correct result, since the majority (e.g., 2 of 3 versions) is correct. If a hardware component and the software component running on it both have errors, the system can again continue to correctly function, if all the other hardware and software components are functioning properly. If less than $\lceil N/2 \rceil$ software or hardware components behave correctly, the system may fail, since the majority does not produce the correct result. The Fault-Tolerant Processor-Attached Processor (FTP-AP) architecture proposed in [29] may be seen as an implementation of this hardware–software fault-tolerant architecture. A quadruple configuration of a core architecture is used as a support for the management of the execution of four diversified software modules running on four distinct processors. *N*-version programming has been exploited in many industrial applications, such as the NASA Space Shuttle [30], the Airbus A320/A330/A340 [31] and Boeing 777 aircraft control systems [32] and various railway signaling and control systems [33–35].

Another major evolution of hardware and software fault-tolerance has been the recovery block approach [26], which consists of three software elements:

- A *primary* module which normally executes the critical software function;
- An *acceptance test* which checks the outputs for correctness;
- An *alternate*[1] module which performs the same function as the primary module, and is invoked by the acceptance test upon detection of a failure in the primary module.

[1]The term *alternate* reflects sequential execution, which is a feature specific to the recovery block approach.

In this approach, the above elements are organized in a manner similar to the passive dynamic redundancy (*standby sparing*) technique adopted for the hardware fault tolerance. The recovery block approach attempts to prevent software faults from impacting on the system environment, and it is aimed at providing fault-tolerant functional components which may be nested within a sequential program. On entry to a recovery block, the state of the system must be saved to permit rollback error recovery. RB performs run-time software, as well as hardware, error detection by applying the acceptance test to the outcome delivered by the primary alternate. If the acceptance test is passed, the outcome is regarded as successful and the recovery block can be exited, discarding the information on the state of the system taken on entry. However, if the acceptance test is not passed (or if any errors are detected by other means during the execution of an alternate), recovery is implemented by state restoration: the system rolls back and starts executing an alternate module from the previously established correct intermediate point or system state, known as *recovery point*. Recovery is considered complete when the acceptance test is passed or all the modules are exhausted. If all the alternates either fail the test or result in an exception (due to an internal error being detected), a failure exception is signaled to the environment of the recovery block. Since the recovery block can be nested, then the raising of such an exception from an inner recovery block would invoke recovery in the enclosing block. In general, multiple alternate procedures can be used. Each procedure must be deliberately designed to be as independent as possible, so as to minimize the probability of having correlated errors in the primary and in the alternate modules. This may be achieved by enforcing design diversity with independently written program specifications, different program languages, algorithms, etc. The acceptance test must be simple, otherwise there will be a significant chance that it contains a fault itself, and so either fails to detect some errors, or falsely identifies some good conditions as being erroneous. Moreover, the test introduces a run-time overhead which could be unacceptable. A number of possible methods for designing acceptance test have been proposed (more details can be found in [9]), but none has been defined as the golden method. Generally speaking, the application test is dependent on the application. As an example, in [36] Algorithm-Based Fault Tolerance (ABFT), error detection techniques are exploited to provide cheap and effective acceptance tests. ABFT has been used in numerical processing for the detection of errors. The ABFT technique provides a transparent error checking method embedded into the functional procedure that can be effectively applied in a recovery block scheme. This method can be applied whenever ABFT is applicable. Indeed, in many real-time applications, the majority of which involve control systems, the numerical processing involved can be adapted to an ABFT solution. Although each of the alternates within a recovery block has to satisfy the same acceptance test, there is no requirement that they all must produce the same results. The only constraint is that the results must be acceptable as determined by the test. Thus, while the primary alternate should attempt to produce the desired outcome, the further alternate may only attempt to provide a degraded service. This is particularly useful in real-time systems, since there may be insufficient time available for complete

functional alternates to be executed when a fault is encountered. The extreme case corresponds to a recovery block which contains a primary module and a null alternate. Under these conditions, the role of the recovery block is simply to detect and recover from errors. In the normal (and most likely case), only the primary alternate of the recovery block is executed as well as the acceptance test, and the run-time overhead of the recovery block is kept to a minimum.

9.4.2 Checkpointing

Checkpointing is a commonly used technique for reducing the execution time for long-running programs in the presence of failures. With checkpointing, the status of the program under execution is saved intermittently in a reliable storage. Upon the occurrence of a failure, the program execution is restarted from the most recent checkpoint rather than from the beginning.

In checkpointing schemes, the task is divided into *n* intervals. At the end of each interval, a checkpoint is added either by the programmer [37] or by the compiler [38, 39]. Fault detection is obtained exploiting hardware redundancy by duplicating the task into two or more processors and comparing the states of the processors at the checkpoints. The probability of two faults resulting in identical states is negligible, so that two matching states indicate a correct execution. By saving at each checkpoint, the state of the task in a reliable storage, the need to restart the task after each fault is avoided.[2] Instead, the task can be rolled back to the last correct checkpoint, and execution resumed from there, thereby shortening fault recovery. Reducing the task execution time is very important in many applications like real-time systems, with hard deadlines, and transactions systems, where high availability is required.

Different recovery techniques can be used to shorten the fault recovery time:

- *Rollback recovery* [37]: both processors are set back to the state of the last checkpoint and the processing interval is retried. If two equal states are reached afterwards, the processing is continued
- *Stop and retry recovery* [37]: if a state comparison mismatches, both processors are stopped until a third processor computes a third status for the mismatching round. Then a two-out-of-three decision is made to identify the fault free version that is used to continue duplex processing
- *Roll-forward checkpoint* [40]: if a state comparison mismatches, the two different states are both stored. The state at the preceding checkpoint, where both

[2]Task duplication [40] was introduced to detect transient faults, based on duplicating the computation of a task on two processors. If the results of the two executions do not match, the task is executed again in another processor until a pair of processors produces identical results. This scheme does not use checkpoints, and every time a fault is detected, the task has to be started from its beginning.

processing modules had agreed, is loaded into a spare module and the checkpoint interval is retried on the spare module. Concurrently, the task continues forward on the two active modules, beyond the checkpoint where the disagreement occurred. At the next checkpoint, the state of the spare module is compared with the stored states of the two active modules. The active module, which disagrees with the spare module, is identified to be faulty and its state is restored to the correct one by copying the state from the other active module, which is fault free. The spare is released after recovery is completed. The spare can be shared among many processor pairs and used temporarily when fault occurs.

In checkpointing schemes, a checkpointing overhead is introduced due to the time to store the processors' states and the time to compare these states. The time spent for compare and store operations may vary significantly, depending on the system, and thus the checkpointing overhead is determined mainly by the operation that takes a longer time. As an example, in a cluster of workstations connected by a LAN, the bandwidth of the communication subsystem is usually lower than the bandwidth of the local storage subsystem. On the other hand, in multiprocessor supercomputers without local disks at the computing nodes, the bandwidth of the communication subsystem is usually higher than the bandwidth of the local storage subsystem.

Different methods have been proposed to reduce checkpointing overhead. The first method is to tune the scheme to the specific system that is implemented on, and use both the compare and the state operations efficiently [41]. Using two types of checkpoint (compare-checkpoints and store-checkpoints) allows tuning the scheme to the system. The compare-checkpoints are used to compare the states of the processors without storing them, while in the store-checkpoints, the processors store their states without comparison. Using two types of checkpoints enables choosing different frequencies for the two checkpoint operations, and utilizing both operations in an efficient way. When the checkpoints that are associated with the operation that takes less time are used more frequently than the checkpoints associated with the operation that takes more time, the recovery time after fault can be reduced without increasing the checkpoint overhead. This leads to a significant reduction in the average execution time of a task. The second method is to reduce the comparison time by using signatures [40], instead of comparing the whole states of the processors. In systems with high comparison time, signatures can significantly reduce the checkpoint overhead, and hence reduce the execution time of a task.

The tradeoffs involved in choosing an appropriate checkpoint frequency are the following. Very frequent checkpoints may cause high overhead due to checkpointing durations, while too rare checkpoints cause longer fault latency and may cause a more probable failure. The effects of varied check intervals and checkpoint periods have been studied in [41]. A main result from that study is that shortening test intervals improves dependability, because the likeliness of two processes affected by a fault is decreased. Thus, it is advised to test states more often than saving checkpoints.

9.4.3 Algorithm-Based Fault Tolerance

The technique presented by Huang and Abraham [42] is aimed at hardening processors when executing matrix applications, such as multiplication, inversion, and LU decomposition. Hardening is obtained by adding coding information to matrices; however, while other approaches introduce coding information (to detect and possibly correct errors) to each byte or word, these coding information are added to whole data structures (in this case to each matrix) or, according to the authors definition, at the *algorithm level*. In the following, the application of ABFT to matrix multiplication will be discussed.

The algorithm is based on modifying the matrices the application is working on according to the definitions introduced in the following.

Definition 9.1. *Given a matrix A composed of $n \times m$ elements, the corresponding column checksum matrix A_c is an $(n + 1) \times m$ matrix, which consists of the matrix A in the first n rows and a column summation vector in the $(n + 1)$th row. Each element of the column summation vector corresponds to the sum of the elements of the corresponding column.*

Definition 9.2. *Given a matrix A composed of $n \times m$ elements, the corresponding row checksum matrix A_r is an $n \times (m + 1)$ matrix, which consists of the matrix A in the first m columns and a row summation vector in the $(m + 1)$th column. Each element of the row summation vector corresponds to the sum of the elements of the corresponding row.*

Definition 9.3. *Given a matrix A composed of $n \times m$ elements, the corresponding full checksum matrix A_f is an $(n + 1) \times (m + 1)$ matrix, which is the column checksum matrix of the row checksum matrix A_r of A.*

The technique proposed in [42] is based on the observation that some matrix operations (matrix by matrix multiplication, LU decomposition, addition, matrix by scalar multiplication, and transposition) preserve the checksum property, according to the following theorems (whose proof can be found in [42])

Theorem 9.1. *When multiplying a column checksum matrix A_c by a row checksum matrix B_r, the result is a full checksum matrix C_f. Moreover, the following relation holds among the corresponding information matrices: $A * B = C$.*

Theorem 9.2. *When a matrix C is LU decomposable, the corresponding full checksum matrix C_f can be decomposed into a column checksum lower matrix and a row checksum upper matrix.*

Theorem 9.3. *When adding two full checksum matrices A_f and B_f, the result is a full checksum matrix C_f. Moreover, the following relation holds among the corresponding information matrices: $A + B = C$.*

Theorem 9.4. *The product of a full checksum matrix and a scalar value is a full checksum matrix.*

Theorem 9.5. *The transpose of a full checksum matrix is a full checksum matrix.*

In order to harden an application performing a matrix operation, one can therefore proceed as follows:

- The operand matrices are transformed into the corresponding row, column, or full checksum matrices, depending on the operation.
- The operation is performed on the checksum matrices.
- A check is performed to detect possible errors, corresponding to the following steps:
  - The sum of all the elements on each row and column is computed.
  - The resulting value is compared with that stored in the row or column summation vector; if a difference is observed, an error is detected.

If we assume that the detected error affected a single element in the result matrix, the identification of the affected element can be performed resorting to the following sequence of operations:

- If a mismatch on both a row and a column summation vector element is detected, the error affected the information element at the intersection of the inconsistent row and column.
- If a mismatch is detected on a row or column summation vector element only, the error affected the summation vector.

After the identification of the faulty element, its correction can be performed resorting to the following sequence of operations:

- If the error affected an information element, the error can be corrected by computing its fault-free value subtracting the sum of the values of the other elements on the same row or column from the corresponding element in the row or column summation vector.
- If the error affected an element of a row or column summation vector, the fault-free value of element can be computed by adding all the elements of the row or column.

It is important to note that in the case of matrices composed of floating point elements, round-off errors could create problems to comparison operations. In this case, some false alarms could be raised. A method to compute the thresholds to be used for distinguishing between round-off errors and errors stemming from faults is outlined in [43] for a similar case.

The ABFT technique is particularly attracting because it introduces a memory and performance overhead that, when compared with other techniques (e.g., TMR), is relatively limited. The introduced memory overhead mainly corresponds to an additional row and column, and it grows linearly with the matrix size as $O(N)$, but the percentage overhead decreases when the matrix size increases, because the memory size grows as $O(N^2)$.

The error detection and correction capabilities of the method are very high when faults affecting the matrices elements during the computation are considered. The method is able to detect and correct any error affecting a single element in the final

matrix. On the other side, the correction capabilities are limited if an error affects more than one element in the resulting matrix.

On the other side, the method is rather weak in detecting and correcting other kinds of faults, e.g., those affecting the memory elements in a microprocessor control unit. If errors in the application code are considered, the method shows some detection capabilities, corresponding to data alterations, although it is definitely unable to detect all the errors of this category.

9.4.4 Duplication

The method first proposed in [44] and then fully described in [45] extends the one proposed by the same authors in [3], in such a way that not only detection, but also fault tolerance is achieved.

The method focuses on computing-intensive applications. Therefore, it is assumed that the program to be hardened begins with an *initialization phase*, during which the data to be elaborated are acquired. This phase is then followed by a *data manipulation phase*, where an algorithm is executed over the acquired data. At the end of the computation, the computed results are committed to the program user, through a *result presentation phase*. The proposed code transformation rules are meant to be applied on the algorithm executed during the data manipulation phase.

The approach exploits code transformation rules providing fault detection and, for most cases, fault correction. The rules are intended for being automatically applied to the program source high-level code and can be classified in two broad categories: rules for detecting and correcting faults affecting data and rules for detecting and (when possible) correcting faults affecting code.

9.4.4.1 Detecting and Correcting Transient Faults Affecting Data

Data hardening is performed according to the following rules:

- Every variable x must be duplicated: let x_0 and x_1 be the names of the two copies. Every write operation performed on x must be performed on x_0 and x_1. Two sets of variables are thus obtained, the former (set 0) holding all the variables with footer 0 and the latter (set 1) holding all the variables with footer 1.
- After each read operation on x, the two copies x_0 and x_1 must be checked, and if an inconsistency is detected, a recovery procedure is activated.
- One checksum c associated to one set of variables is defined. The initial value of the checksum is equal to the exor of all the already initialized variables in the associated set.
- Before every write operation on x, the checksum is re-computed, thus canceling the previous value of x ($c' = c \wedge x_0$).
- After every write operation on x, the checksum is updated with the new value x' ($c' = c \wedge x'_0$).

The recovery procedure re-computes the exor on the set of variables associated to the checksum (set 0, for example), and compares it with the stored one. Then, if the re-computed checksum matches the stored one, the associated set of variables is copied over the other one; otherwise, the second set is copied over the first one (e.g., set 0 is copied over set 1, otherwise set 1 is copied over set 0).

9.4.4.2 Detecting and Correcting Transient Faults Affecting the Code

To detect faults affecting the code, the method exploits the techniques introduced in [20]. The first technique consists in executing any operation twice, and then verifying the coherency of the resulting execution flow. Since most operations are already duplicated due to the application of the rules described in the previous subsection, this idea mainly requires the duplication of the jump instructions. In the case of conditional statements, this can be accomplished by repeating twice the evaluation of the condition.

The second technique aims at detecting those faults modifying the code so that incorrect jumps are executed, resulting in a faulty execution flow. This is obtained by associating an identifier to each *basic block* in the code. An additional instruction is added at the beginning of each block of instructions. The added instruction writes the identifier associated to the block in an ad hoc variable, whose value is then checked for consistency at the end of the block.

The recovery procedure consists in a rollback scheme: as soon as a fault affecting the program execution flow is detected, the program is restarted (i.e., the program execution is restarted from the data manipulation phase, or from a safe point which has been previously recorded). Thanks to this solution, it is possible to:

- Detect and correct transient faults located in the processor internal memory elements (e.g., program counter, stack pointer, and stack memory elements) that temporarily modify the program execution flow.
- Detect transient faults originated in the processor code segment (where the program binary code is stored) that permanently modify the program execution flow. As soon as a fault affects the program code memory, the bit-flip it produces is indeed permanently stored in the memory, causing permanent modification to the program binary code. Restarting the program execution when such a kind of fault is detected is insufficient for removing the fault from the system. As a result, the program enters in an end-less loop, since it is restarted every time the fault is detected. This situation can be easily identified by a watch-dog timer that monitors the program operations.

9.4.4.3 Duplication and Hamming Code

The method (which was first proposed in [46]) exploits the properties of the error correcting codes for achieving fault-tolerance. Error correcting code introduces

information redundancy into the source code. These transformations, which were automatically performed, introduce data and code devoted to detect, and eventually correct, possible errors corrupting information stored in the memory area.

The basic idea is to associate an extra code information to every variable in the code. This extra code (*Hamming corrector*) is computed according to Hamming codes. This code is able to correct a single error and detect a double error. A detailed description of how this code is computed is out of the scope of this chapter.

Every time a variable is modified, its correspondent Hamming corrector has to be updated. On the other hand, in a read operation, the Hamming corrector is used for the decoding operation. Two parameters are needed for this operation: the variable's value and the variable's Hamming corrector code. In the case of the corruption of one of these two values, the decoding procedure will take one of the possible following decisions:

- If one bit is corrupted, the decision is a correction of the corrupted bit.
- It two bits are damaged, then the decision is the detection, without correction possibility.
- If more than two bits are affected, the decision is an erroneous correction.

Experimental results demonstrate the feasibility of the approach and its detection and correction capability. As far as fault affecting data are considered, 0.7% of faults produce a failure, but 36% of faults are detected and 32% of faults are detected and corrected. As far as faults affecting code are considered, 3% of faults produce a failure, but 53% of them are detected, and 0.2% are detected and corrected.

The major drawback of error detection and correction methods based in Hamming codes comes from the resulting memory area overhead (due to hamming corrector codes and decoding operations) and the increase in execution time due to hamming corrector code update and decoding operation execution. The overhead factors obtained considering a benchmark program corresponds to an execution time 12 times higher than the one required for the unhardened program and a memory size 3 times higher than the one for the original unhardened program. These overhead factors show that this method can only be applied when the fault tolerance requirements justify high overhead penalty.

9.5 Conclusions

In the last years, both industry and academia devoted significant resources to explore the area of software-level mitigation techniques, with the goal of developing solutions that are able to cope effectively with the effects of permanent and temporary faults, while matching the strict design and cost constraints existing in areas such as automotive, biomedical, and telecom. Several techniques have been proposed, which have been overviewed in this chapter, and some of them have been already adopted for products deployed on the field. Interestingly, these techniques

are now used even for products working in harsh environments (such as the space one), thus also proving their flexibility.

References

1. M. Rebaudengo, M. Sonza Reorda, M. Torchiano, M. Violante, Soft-error detection through software fault-tolerance techniques. Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 1999, pp. 210–218
2. M. Rebaudengo, M. Sonza Reorda, M. Torchiano, M. Violante, A source-to-source compiler for generating dependable software. Proceedings of the IEEE International Workshop on Source Code Analysis and Manipulation, 2001, pp. 33–42
3. P. Cheynet, B. Nicolescu, R. Velazco, M. Rebaudengo, M. Sonza Reorda, M. Violante, Experimentally evaluating an automatic approach for generating safety-critical software with respect to transient errors. IEEE Transactions on Nuclear Science 47(6), 2000, 2231–2236
4. A. Benso, S. Chiusano, P. Prinetto, L. Tagliaferri, A C/C++ source-to-source compiler for dependable applications. Proceedings of the IEEE International Conference on Dependable Systems and Networks, 2000, pp. 71–78
5. N. Oh, P.P. Shirvani, E.J. McCluskey, Error detection by duplicated instructions in super-scalar processors. IEEE Transactions on Reliability 51(1), 2002, 63–75
6. G. Sohi, M. Franklin, K. Saluja, A study of time-redundant fault tolerance techniques for high-performance pipelined computers. 19th International Symposium on Fault Tolerant Computing, 1989, pp. 463–443
7. C. Bolchini, A software methodology for detecting hardware faults in VLIW data paths. IEEE Transactions on Reliability 52(4), 2003, 458–468
8. N. Oh, E.J. McCluskey, Error detection by selective procedure call duplication for low energy consumption. IEEE Transactions on Reliability 51(4), 2002, 392–402
9. K. Echtle, B. Hinz, T. Nikolov, On hardware fault detection by diverse software. Proceedings of the 13th International Conference on Fault-Tolerant Systems and Diagnostics, 1990, pp. 362–367
10. H. Engel, Data flow transformations to detect results which are corrupted by hardware faults. Proceedings of the IEEE High-Assurance System Engineering Workshop, 1997, pp. 279–285
11. M. Jochim, Detecting processor hardware faults by means of automatically generated virtual duplex systems. Proceedings of the International Conference on Dependable Systems and Networks, 2002, pp. 399–408
12. S.K. Reinhardt, S.S. Mukherjee, Transient fault detection via simultaneous multithreading. Proceedings of the 27th International Symposium on Computer Architecture, 2000, pp. 25–36
13. E. Rotenberg, AR-SMT: a microarchitectural approach to fault tolerance in microprocessors. 29th International Symposium on Fault-Tolerant Computing, 1999, pp. 84–91
14. N. Oh, S. Mitra, E.J. McCluskey, ED4I: error detection by diverse data and duplicated instructions. IEEE Transactions on Computers 51(2), 2002, 180–199
15. M. Hiller, Executable assertions for detecting data errors in embedded control systems. Proceedings of the IEEE International Conference on Dependable Systems and Networks, 2000, pp. 24–33
16. J. Vinter, J. Aidemark, P. Folkesson, J. Karlsson, Reducing critical failures for control algorithms using executable assertions and best effort recovery. Proceedings of the IEEE International Conference on Dependable Systems and Networks, 2001, pp. 347–356
17. S.S. Yau, F.-C. Chen, An approach to concurrent control flow checking. IEEE Transactions on Software Engineering 6(2), 1980, 126–137
18. N. Oh, P.P. Shirvani, E.J. McCluskey, Control-flow checking by software signatures. IEEE Transactions on Reliability 51(2), 2002, 111–122

19. Z. Alkhalifa, V.S.S. Nair, N. Krishnamurthy, J.A. Abraham, Design and evaluation of system-level checks for on-line control flow error detection. IEEE Transactions on Parallel and Distributed Systems 10(6), 1999, 627–641
20. O. Goloubeva, M. Rebaudengo, M. Sonza Reorda, M. Violante, Soft-error detection using control flow assertions. Proceedings of the 18th International Symposium on Defect and Fault Tolerance in VLSI Systems, 3–5 November 2003, pp. 581–588
21. R. Vemu, J.A. Abraham, CEDA: control-flow error detection through assertions. Proceedings of the 12th IEEE International On-Line Testing Symposium, 2006, pp. 151–158
22. R. Vemu, J.A. Abraham, Budget-dependent control-flow error detection. Proceedings of the 14th IEEE International On-Line Testing Symposium, 2008, pp. 73–78
23. C. Babbage, On the mathematical powers of the calculating engine, unpublished manuscript, December 1837, Oxford, Buxton Ms7, Museum of History of Science. Printed in *The Origins of Digital Computers: Selected Papers*, B. Randell (ed.), Springer, Berlin, 1974, pp. 17–52
24. A. Avizienis, J.C. Laprie, Dependable computing: from concepts to design diversity. Proceedings of the IEEE 74(5), 1986, 629–638
25. A. Avizienis, The N-version approach to fault-tolerant software. IEEE Transactions on Software Engineering 11(12), 1985, 1491–1501
26. B. Randell, System structure for software fault tolerance. IEEE Transactions on Software Engineering 1(2), 1975, 220–232
27. D. Pradhan, *Fault-Tolerant Computer System Design*. Prentice-Hall, Englewood Cliffs, NJ, 1996
28. J.P. Kelly, T.I. McVittie, W.I. Yamamoto, Implementing design diversity to achieve fault tolerance. IEEE Software 8(4), 1991, 61–71
29. J.H. Lala, L.S. Alger, Hardware and software fault tolerance: a unified architectural approach. Proceedings of the 18th International Symposium on Fault-Tolerant Computing, 1988, pp. 240–245
30. C.E. Price, Fault tolerant avionics for the space shuttle. Proceedings of the 10th IEEE/AIAA Digital Avionics Systems Conference, 1991, pp. 203–206
31. D. Briere, P. Traverse, AIRBUS A320/A330/A340 electrical flight controls: a family of fault-tolerant systems. Proceedings of the 23rd International Symposium on Fault-Tolerant Computing, 1993, pp. 616–623
32. R. Riter, Modeling and testing a critical fault-tolerant multi-process system. Proceedings of the 25th International Symposium on Fault-Tolerant Computing, 1995, pp. 516–521
33. G. Hagelin, ERICSSON safety system for railway control. Proceedings of the Workshop on Design Diversity in Action, Springer, Vienna, 1988, pp. 11–21
34. H. Kanzt, C. Koza, The ELEKTRA railway signalling system: field experience with an actively replicated system with diversity. Proceedings of the 25th International Symposium on Fault-Tolerant Computing, 1995, pp. 453–458
35. A. Amendola, L. Impagliazzo, P. Marmo, G. Mongardi, G. Sartore, Architecture and safety requirements of the ACC railway interlocking system. Proceedings of IEEE International Computer Performance and Dependability Symposium, 1996, pp. 21–29
36. A.M. Tyrrell, Recovery blocks and algorithm-based fault tolerance, EUROMICRO 96. Beyond 2000: Hardware and Software Design Strategies. Proceedings of the 22nd EuroMicro Conference, 1996, pp. 292–299
37. K.M. Chandy, C.V. Ramamoorthy, Rollback and recovery strategies for computer programs. IEEE Transactions on Computers 21(6), 1972, 546–556
38. W.K. Fuchs, C.-C.J. Li, CATCH – compiler-assisted techniques for checkpointing. Proceedings of the 20th Fault-Tolerant Computing Symposium, 1990, pp. 74–81
39. J. Long, W.K. Fuchs, J.A. Abraham, Compiler-assisted static checkpoint insertion. Proceedings of the 22nd Fault-Tolerant Computing Symposium, 1992, pp. 58–65
40. D.K. Pradhan, N.H. Vaidya, Roll-forward checkpointing scheme: a novel fault-tolerant architecture. IEEE Transactions on Computers 43(10), 1994, 1163–1174
41. A. Ziv, J. Bruck, Performance optimization of checkpointing scheme with task duplication. IEEE Transactions on Computers 46(12), 1997, 1381–1386

42. K.H. Huang, J.A. Abraham, Algorithm-based fault tolerance for matrix operations. IEEE Transactions on Computers C-33(6), 1984, 518–528
43. A. Roy-Chowdhury, P. Banerjee, Tolerance determination for algorithm based checks using simplified error analysis. Proceedings of the IEEE International Fault Tolerant Computing Symposium, 1993
44. M. Rebaudengo, M. Sonza Reorda, M. Violante, A new software-based technique for low-cost fault-tolerant application. Proceedings of the IEEE Annual Reliability and Maintainability Symposium, 2003, pp. 25–28
45. M. Rebaudengo, M. Sonza Reorda, M. Violante, A new approach to software-implemented fault tolerance. Journal of Electronic Testing: Theory and Applications 20, 2004, 433–437
46. B. Nicolescu, R. Velazco, M. Sonza Reorda, Effectiveness and limitations of various software techniques for "soft error" detection: a comparative study. Proceedings of the IEEE 7th International On-Line Testing Workshop, 2001, pp. 172–177

Chapter 10
Specification and Verification of Soft Error Performance in Reliable Electronic Systems

Allan L. Silburt, Adrian Evans, Ana Burghelea, Shi-Jie Wen, David Ward, Ron Norrish, Dean Hogle, and Ian Perryman

This chapter describes the modeling, analysis, and verification methods used to achieve a reliability target set for transient outages in equipment used to build the backbone routing infrastructure of the Internet. We focus on ASIC design and analysis techniques that were undertaken to achieve the targeted behavior using the 65-nm technology. Considerable attention is paid to Single Event Upset in flip-flops and their potential to produce network impacting events that are not systematically detected and controlled. Using random fault injection in large-scale RTL simulations, and slack time distributions from static timing analysis, estimates of functional and temporal soft error masking effects were applied to a system soft error model to drive decisions on interventions such as the choice of flip-flops, parity protection of registers groupings, and designed responses to detected upsets. Central to the design process is a modeling framework that accounts for the upset effects and relates them to the target specification. This enables the final system to be tested using large area neutron beam radiation to confirm the specification has been met.

A.L. Silburt (✉), A. Evans (✉), and A. Burghelea
Cisco Systems, SSE Silicon Group
e-mail: asilburt@cisco.com

S.-J. Wen
Cisco Systems, Advanced Manufacturing Technology Centre

D. Ward
Juniper Networks

R. Norrish
Cisco Systems, Technical Failure Analysis

D. Hogle
Cisco Systems, SSE CRS Hardware

I. Perryman
Ian Perryman & Associates

M. Nicolaidis (ed.), *Soft Errors in Modern Electronic Systems*,
Frontiers in Electronic Testing 41, DOI 10.1007/978-1-4419-6993-4_10,
© Springer Science+Business Media, LLC 2011

10.1 Introduction

Earlier chapters of the book have dealt with methods to model, measure and mitigate soft error effects in electronic systems. This chapter deals with the challenge of building a highly reliable ASIC intensive product where the mitigation strategies applied at the most fundamental circuit level combine to achieve a desired outcome for system performance. There is a cost associated with the various circuit level strategies in terms of silicon area, power, and increased design complexity. Decisions on when and where to use ECC (Error Correcting Code) protection on memories, to replicate logic or to use larger, SEU resistant flip-flops must be made during chip design, years ahead of the system's deployment. Architectural features such as localized reset islands, table scrubs, and other techniques involving software and hardware cooperation must also be designed in up-front. At this early stage, experimental data on SEU rates for many of the technologies are often lacking since they are in parallel development. All of these factors contribute to an uncertain decision-making process regarding which features must be implemented to achieve the end result of a customer experience of reliability that meets their expectations. This chapter defines a comprehensive specification, modeling, and experimental verification methodology for the soft error performance of a hardware assembly (referred to as a FRU – Field Replaceable Unit) in an Internet core router system. It can be applied to any integrated system that is required to meet a particular high availability target and must link design tradeoffs at the ASIC and system design phase to end-user experience.

10.2 System Soft Error Specification

System soft error performance analysis requires a convergence of failure rate metrics and outage impact. Failure rates are measured in FIT (Failures in Time) defined by

$$\begin{aligned} 1\,\text{FIT} &= 1\,\text{Failure}/10^9\ \text{Hours} \\ &= 10^9/\text{MTBF}, \end{aligned} \tag{10.1}$$

where the MTBF (Mean Time Between Failures) is measured in hours. It is most common to see component hard and soft failure rates in FIT and permanent failure rate of hardware assemblies in units of MTBF.

System outages are measured in DPM (Defects per million) where for a given time period,

$$\text{DPM} = 10^6 \times \left(\frac{\sum \text{outage durations}}{\text{period\_duration}} \right) \tag{10.2}$$

The popular term "5 9's" availability refers to 99.999% uptime and translates to 10 DPM. For a given component, c, with a soft failure rate of f_c FIT, and each failure causing an outage of t_c seconds, its DPM contribution would be

$$\mathrm{DPM_c} = \frac{t_c \times f_c}{3.6 \times 10^6}. \tag{10.3}$$

DPM is a convenient measure to work with when analyzing network reliability, since independent component contributions to outage can be simply added together.

Quality of Service (QoS) for a particular Internet application such as video conferencing is typically quoted in Packet Loss Ratio (PLR). Under the assumption of full bandwidth utilization by the target application, this is related to DPM according to

$$\mathrm{PLR} = \mathrm{DPM} \times 10^{-6} \tag{10.4}$$

10.2.1 Internet Core Network Requirements

When looking at a hardware assembly from an Internet service provider perspective, they will view their network elements against three independent expectations. The first is the general service availability of the network in the order of 10 DPM. This is targeted to meet service level agreements that network providers commit to their clients. The second is a hard (permanent) failure rate in the order of 100,000 h MTBF that is targeted toward their own network operating costs (sparing ratios, maintenance contracts, cost amortization periods, etc.). The third is the QoS requirement for the most demanding target application for which the network is being designed. Although each of these three expectations could be expressed in the same units, they rarely are. For example, the hardware failure rate target of 100,000 h MTBF could be represented as an availability target by factoring in the time to replace or route around a failed unit. What distinguishes each of these is the scale in time in which the network is affected. The 10 DPM service availability target is a long-term average over which a client is expected never to be denied access to the network. The QoS specification refers to very brief outages that may affect user experience in a particular session. The hardware failure rate is a long-term average occurrence rate which factors into an operations and maintenance cost for the equipment that would only be apparent to a user through their service fees.

Outages due to soft errors do not fit cleanly into any of these targets. When a soft error causes a larger outage, forcing a line card or system reboot time of many minutes, for example, it may become indistinguishable from a hardware failure from a network operator's perspective. In these cases, diagnostic information may be insufficient or at best non-persuasive in proving that there is no underlying

hardware fault, perhaps intermittent, that caused such an uncontrolled outage. So, the response is often to replace the unit and consider the event a contribution to the hard failure rate. On the contrary, a controlled outage of a few seconds' duration and good diagnostic presentation would be considered only to contribute to general network availability and not result in any hardware replacement. An outage in the range of a few milliseconds is only relevant to a QoS target since the session may not be lost, yet the end-user experience may be affected.

A framework for tying these requirements together is illustrated in Fig. 10.1. This chart plots error rate against outage time. The vertical axis is calibrated both in FIT and in outage frequency for a group of 100 units to represent what a service provider might observe in monitoring a large network. A line representing 10 DPM illustrates a set of operating points on this graph that would satisfy the 5 "9"'s requirement if this network element were the only contributor to network outage. Other lines showing 0.0001 and 1 DPM are shown and discussed further below.

The general approach to this specification is to categorize failure mechanisms according to the class of outage they generate in a working system. Binning in this way allows us to define a FIT target for each of the categories well enough to make design decisions and later to verify performance. The horizontal axis shows the outage time in seconds, and groups the axis into four regions that classify the severity of the outage as "low", "medium", "high", and "unrecoverable" where the latter is an undetected/unbounded outage. An additional category "no outage" is effectively compressed into left hand vertical axis to account for soft errors that are corrected with zero impact. Individual hardware components can be placed on this chart based on available soft error FIT rate data and the response of the system to

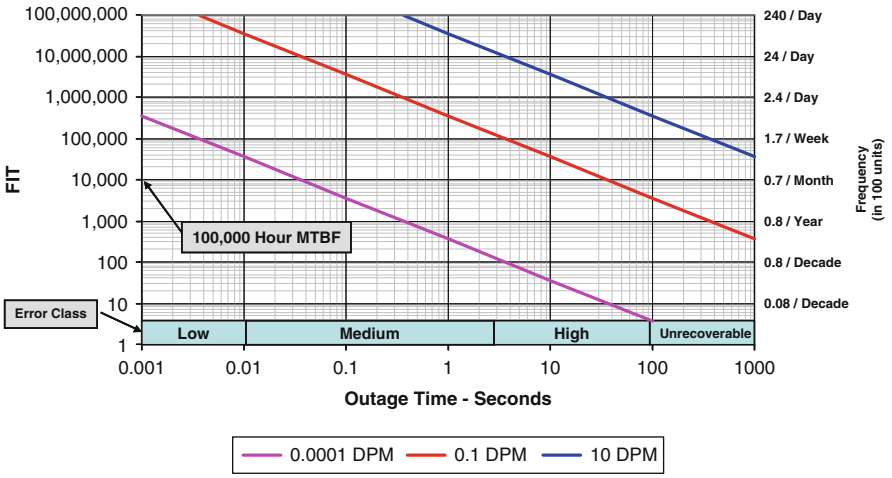


Fig. 10.1 FIT rate versus recovery time performance metric chart. The vertical axis is the aggregated component FIT rate for a hardware assembly and the horizontal axis is outage time associated with a transient fault

upset events. Single bit errors (SBE), multiple bit errors (MBE), single event latchup, and hard failures are each considered so that each component would contribute a unique point for each failure mechanism. Summing up all of the contributing effects for each category constitutes a model for the transient outage performance of the card.

The outage categories defined on the chart are large – each covering an order of magnitude or more in outage time. Experience has shown that this is sufficient since outage times are not predictable with great precision during the design phase and will vary in the final product depending on how the router is configured in a network. What is important is to target a maximum tolerable outage time for a class of effects so that the appropriate response can be designed. Hence, it is the outage time thresholds separating the regions that are important design targets. These thresholds are intended to capture the impact of an outage on the network operator as well as an end-user's experience. It should be noted that the outage effect on service availability may be a loss of packets or a fault in how packets are routed, classified, prioritized, or counted. Only the duration is specified and for the purposes of this chapter we will consider all effects to be simply an outage. This conservative assumption simplifies the model considerably. In addition, an outage is assumed to affect 100% of the traffic passing through the unit. If the impact of an error is less than the full bandwidth, the FIT contribution for that element is scaled down accordingly.

The "low-severity" category is defined as an outage that has no impact on any routing protocol and has an impact to an end-user voice, video, or data application that is imperceptible. In 2002, the Internet2 QoS working group [1] surveyed several sources and concluded that delays of less than 150 ms in Voice over IP sessions were not perceptible and that up to 2% packet loss is acceptable when the 150-ms delay target is met. In 2003, the International Telecommunications Union published Recommendation G.114 [2] which also illustrated 150 ms "mouth-to-ear" delay as a critical target not to exceed for interactive voice applications. The study of human perception of video quality is complex and the existence of different video standards, compression algorithms, and distribution protocols make the choice of a single number difficult. Some early informal experiments at our laboratories indicated that random drops inserted into MPEG streams became impossible to detect at around 150 ms in duration, which was surprisingly consistent with the above specifications for voice and may have to do with basic human perceptual processing time.

In addition to human perception thresholds, data networks also have their own rhythms that a router must support. Connections across an IP network are managed by several layers of protocols that monitor the health of the connection path. These protocols are designed to detect and respond to failures in any segment by re-transmitting or re-routing traffic through alternate paths. As IP-based data networks are being constructed for ever more demanding applications in finance and high-quality video distribution, faster response to network outages are required. The Bidirectional Forwarding Distribution (BFD) protocol [3] that operates between line cards on adjacent hops across an IP network represents the most

demanding requirement. BFD timers may be set as low as 10 ms in next generation equipment deployments. Hence, the threshold for avoidance of any outage detection for the "low-severity" category is 10 ms.

The "medium-severity" category covers the range where outages will cause noticeable effects to an end user with increasing severity at the upper end of the range. Since routing protocols are now affected, the network may route around a node that stops forwarding over this interval. These routing changes propagate through the network as adjacent nodes spread the new topology and they are highly visible to the network provider as a "route flap." The upper end of this bin is set at 3 s, which has historically been the threshold for call drop in legacy voice networks. Data file transfers across the Internet that are not time sensitive typically use the Transmission Control Protocol (TCP). This highly resilient method operates between end points and is capable of ensuring high data quality by signaling re-transmission of segments of a file when an error is detected. The Internet Engineering Task Force (IETF) Network Working Group's published recommendation [4] specifies 3 s as the default interval for a TCP session to wait prior to re-transmission. This provides additional support for the 3 s "medium-severity" threshold.

The "high-severity" category begins at 3 s and ends at the time required for the FRU to perform a full reboot and return to active service. This covers outages that have been detected by the hardware but have no controlled response other than a complete restoration of state under software control. The upper bound will be specific to a particular hardware unit. It is distinguished from the "unrecoverable" category which represents an undetected error where the damaged state causes some behavior to persist without the awareness of the system's controller and as such the software does not respond. In Fig. 10.1, the upper bound for "high" is set at 3 min; however, there is considerable uncertainty in this number during the hardware development stage and it will depend substantially on the router configuration, size of tables etc. In practice, this value of this upper bound is not important. The "high-severity" category represents all outages that are detected by the system with recovery time larger than 3 s. The "unrecoverable" category represents all outages that are not detectable by the system and hence may have an unbounded outage time. Without well-defined outage time maximums, it is clear that these categories cannot be specified using a DPM metric. In practice, these two categories are also the most difficult specifications to meet.

In order to achieve demanding targets for network availability, providers will often implement one of several forms of hardware redundancy and supporting mechanism to detect a fault and initiate switchover from the main to the spare. Although this does not change the basic performance of the FRU itself, it does permit a decoupling of end-user experience from some of the outage behaviors described in the specification. For example, an effective mechanism for detecting faults and switching traffic to a hot standby network element in 200 ms allows all detectable faults causing larger outages to be mapped to $x = 200$ ms and summed to the medium-severity category from the point of view of end-user service impact. This replaces, for example, the length of time allocated for a technician to swap a faulty line card with 200 ms in the calculation of service availability impact of the

100,000 h MTBF hardware failure rate. However, it is important to note that for soft errors, there is more to consider. An FRU that experiences an outage due to a soft error must not be replaced since it is not faulty. It is expected to recover and return to active status perhaps as the new hot standby unit and it is desirable that this period of vulnerability to a second fault be as brief as possible. Therefore, when constructing a specification for soft error performance of a hardware unit, it is important to look at its complete behavior under worst case conditions considering that it may be used in applications with and without redundancy.

10.2.2 Constructing the Specification

We construct the soft error performance specification by considering each of the following three reference points:

1. The overall network availability target (10 DPM/ 5“9”’s);
2. The most demanding application’s QOS requirements;
3. The 100,000 h MTBF hardware failure rate.

We also consider that the products built from the chip set under design may be used in stand-alone configurations or in networks that include redundant hardware that can be switched into service based on some detection mechanism.

1. First, we consider the 5 9’s (10 DPM) availability target for an end-to-end service traversing a network. A service provider will have a number of pieces of equipment along a path between hand-off points covered in a service level agreement with a client. We assume that the router is allocated 1/10 of the budget or 1 DPM. Within the router, the 1 DPM must be apportioned to the major outage generating mechanisms in hardware and software. The calculation is made from the perspective of a path transiting through the equipment and all infrastructures that affect that path.

   A simplified diagram of the major hardware components of an Internet core router is shown in Fig. 10.2.

   The Line Card receives the signal from one or several fiber optic links in the network, performs the optical to electrical conversion, parses the data into packets, and makes a number of decisions on routing, traffic management, filtering, statistics gathering, priority, and many other features prior to forwarding packets via the switch fabric. The route processor handles special packets used to manage the network and build a global view of connectivity which can be mapped to each line card and provide supervisory functions for the system. Each of these units can be operated in redundant configurations.

   A path through the router for this analysis would constitute an ingress line card, egress line card, and switch fabric on the data path as well as route processors (control plane) and general infrastructure (chassis, power supply, fans, etc.). We consider the worst case scenario in which there is no line card

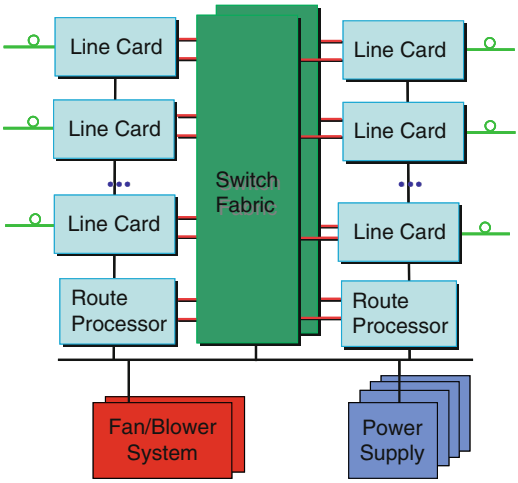


Fig. 10.2 Simplified architecture diagram of an internet core router

redundancy features implemented and split the 1 DPM into contributions from SW failures, HW failures, and soft errors. Assuming for simplicity an equal distribution among these three categories, the soft error share would be approximately 0.3 DPM to split among the major hardware units (line card, route processor, and switch fabric) or roughly 0.1 DPM for a line card. If we consider just the high-severity errors alone, availability in the order of 0.1 DPM would correspond to occurrence rates on the order of 100,000 FIT (~2 occurrences per week in 100 FRUs), which is well beyond the tolerance level of large network operators. Hence, analysis of network availability provides a very weak constraint on soft error performance. The specification must be more strict than that inferred from 5 9's availability.

2. The International Telecommunications Union published recommendation Y.1541 [5] in 2006. This document specifies an end-to-end PLR requirement of 1×10^{-3} and a packet error ratio (PER) of 1×10^{-4} for what was considered the most demanding class of services required for voice over Internet protocol (VoIP) and Video Teleconferencing. A provisional target of PLR and PER $< 1 \times 10^{-5}$ and IPER $< 1 \times 10^{-5}$ was also listed. Using the worst case assumption of 100% bandwidth utilization by this service as noted in Sect. 10.1, this corresponds to 10 DPM which is again a very weak constraint.

   The recent rapid growth of the Internet as a video distribution medium has introduced applications with more demanding QOS targets. These were foreshadowed in Appendix VIII of [5], where PLR as low as 4×10^{-11} was reported but acknowledged to be outside that specification. New protocols such as BFD [3] and improved performance in routing table reconvergence after a fault is discovered have made it possible to build IP networks for high-quality broadcast video distribution. In the absence of a formal ITU-T or IETF QOS requirement, we have adopted a PLR of 1×10^{-10} (one performance hit per day) corresponding to the 0.0001 DPM line in Fig. 10.1. This is most relevant in networks that have

implemented redundancy and a fast detection protocol such as BFD. In this case, all outages in the medium to high-severity category map to the detection/switch-over time interval. In today's systems, this is about 50 ms. This limits the occurrence rate for these outages to 10,000 FIT, which is equivalent to the hardware failure rate which sets the maximum for the medium-severity category. Note that this 0.0001 DPM target is applied to the line card being modeled here and does not capture the end-to-end QOS performance as there are other hops in that path to consider as in the 5 "9"'s analysis above. However, in this case, we are able to take advantage of other network features such as receiver host buffering at end points, say a set top box, which combined with a re-transmit capability, enable the target end-to-end performance to be met.

3. The third constraint on soft error performance is derived from the hard failure rate target of 10,000 FIT (100,000 h MTBF). This represents an independent network operator requirement. Transient outages in the "high-severity" class often result in the unit being replaced as discussed earlier. This sets the floor for tolerable soft error FIT rates for each of the "High" and "Unrecoverable" categories. Permanent component failures follow a "bathtub curve" clustered at the beginning and end of their operating life, whereas soft error transients follow a uniform distribution in time. As a result, during the middle part of the equipment lifetime, the transient error rate will appear more pronounced relative to hard failures. For these reasons, we set the maximum high-severity and undetectable soft failure rate at 10% of the long-term average hardware failure rate or 1,000 FIT. This ensures that hardware field returns with "no fault found" diagnosis and attributable to transient errors never become a significant statistic at our repair depots over the lifetime of the product. It also avoids the situation where we over-design our ASIC soft error performance at the price of component cost and system complexity without delivering any real benefit to customers beyond the hardware MTBF.

   Figure 10.3 shows the target specification derived from these three constraints. The shaded region represents the maximum FIT rate allowed in each of the outage classes. The most demanding requirement is in the "high" and "unrecoverable" classes set at 10% of the target hardware failure rate. The "medium-severity" category is given a target equal to the hardware failure rate. Since a medium-severity outage represents a software managed response, it will be easily distinguished from any hardware issue and can, therefore, be treated independently of the hard failure rate. However, since the effects will be very visible as route flaps, an occurrence rate of more than 10,000 FIT will not be tolerated by network operators. The low-severity outage category is set by the 0.0001 DPM QOS requirement at 30,000 FIT; although considering that outages below 10 ms may not be perceptible in any application, it is difficult to establish a strong justification for the limit of these short disruptions.

   The specification is summarized in Table 10.1 along with the calculated outage contribution in seconds per year for each of the outage categories. Note that the outage time for the "unrecoverable" category is not deterministic. The 1,000 s value in the table is shown to illustrate the sensitivity of the overall soft

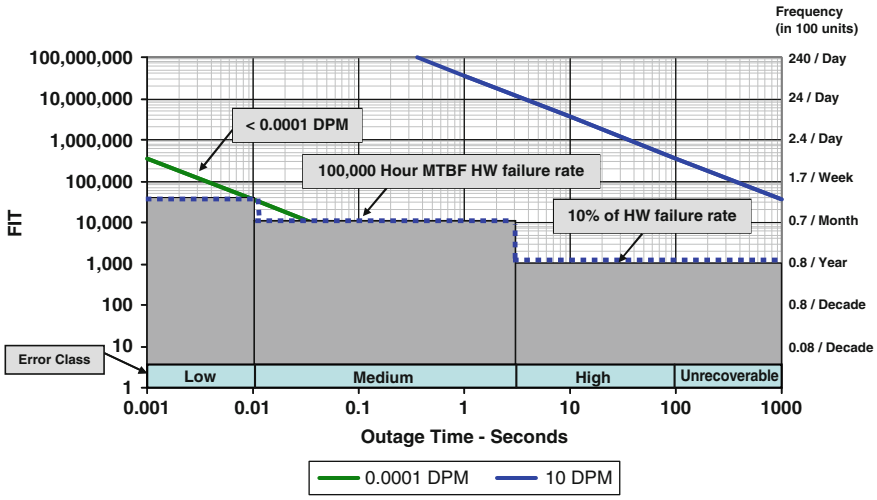


Fig. 10.3 Soft error target specification (*gray section*)

Table 10.1 Soft error target specification summary (single field replaceable unit)

Class	Maximum recovery time (s)	Maximum outage rate (KFIT)	DPM	Maximum (s/year)
No outage	0	Unlimited	0	N/A
Low	0.01	30	0.000	0.00
Medium	3	10	0.008	0.26
High	180	1	0.050	1.58
Unrecoverable	1000	1	0.278	8.77
		Total	0.336	10.61

error performance to this category as well as the "high" category. In practice, these two categories are also the most difficult specifications to meet. As will be discussed later, outages of this type below 1,000 FIT are made up of hundreds of thousands of very small contributors that are, at this time, at the limits of our ability to measure and model with sufficient accuracy.

10.3 Designing a System to Meet the Specification

Designing a hardware assembly and chip set to meet the specification involves building a comprehensive soft error model. This is a bottom-up process that simply counts every instance of every hardware element – typically in a spreadsheet format. A modern line card for an Internet core router contains hundreds of highly integrated silicon components susceptible to soft error effects, including custom ASICs, third party application specific chips, standard microprocessors and micro

Table 10.2 Internet core line card aggregated component single and multi-bit error rates. Raw FIT rate refers to the rate before any de-rating factors have been applied

	Mbits	Raw SBE FIT rate	Raw MBE FIT rate
Embedded SRAM	219	223,190	1,063
External SRAM	370	148,000	0
External DRAM	45,440	45,440	0
TCAM	80	112,000	0
Flip-flops	7	14,205	0
Total	46,116	552,835	1,063

controllers, SRAMs, DRAMs, Ternary Content Addressable Memories (TCAM), and Field Programmable Gate Arrays (FPGA).

For a case study, we will take a specific FRU in a next generation router that has multiple instances of four different ASICs using 65-nm process technologies. A summary of the major soft error producing components by cell type is shown in Table 10.2 along with the single and multi-bit error rates.

The raw single bit soft error rate of these components combined exceeds 550,000 FIT, which is about 5 errors per year per line card or a little over 1 error per day in a 100-card network. It is worth noting that ASIC embedded SRAM contributes the largest share of the FIT rate and that DRAM FIT rates per MBit are so low that DRAM is significant only due to the sheer volume of bits on the card.

All of the single and multi-bit error calculations are based ultimately on data supplied by silicon vendors. The quality of this data can vary substantially and particularly for the second order effects such as flip-floperror rates and SRAM multi-bit error rates. At the time of this writing, most silicon component vendors do not provide data on these second order effects at all and many ASIC vendors, when they can provide it, are lacking measured data to support it.

For our case study, the critical design decisions affecting soft error rates are made in the HW architecture and ASIC micro architectures. For example, the choice of an off-the-shelf microprocessor will bring with it whatever mechanism for cache error correction that was designed in. We include the contributions of the processor but we can do little to change its error rate contributions. Most of the memory on this card will be controlled by ASICs and ASICs will constitute most of the flip-flop-based storage as well. All of the error contributions listed in Table 10.2 must be individually accounted for with a method for detection, correction, software notification as appropriate, or an argument for self-correcting behavior. In addition, a means for injecting the error is required to verify the expected response in the laboratory.

10.3.1 Memories

Extensive application of single error correction double error detection (SECDED) ECC [6] to all SRAMs and DRAMs places 75% of the single bit errors in Table 10.1

in the "no outage" category. Most of the remainder is made up of TCAM single bit errors. Generally, since TCAMs cannot have error correction built into their search operation, TCAM error detection and correction are performed using a background table scrub operation. Embedded hardware performs these scans and raises an interrupt when a parity check fails, and the software must respond and correct the error. A complete scan of the array along with the SW response to repair an error takes on the order of 10–20 ms, which places this in the medium-severity category. However, since a single bit upset in the TCAM impacts less than 1% of the total traffic load on average, we can de-rate this contribution by a factor of 100.

Some embedded SRAM structures are not protected by ECC due to the cost, complexity, or performance impact of the added gates. These are exceptions that are justified using detailed analysis that traces the impact of an error and proves that it is benign. For example, data structures traversing a chip may have an end-to-end error detection mechanism (e.g., CRC and Reed-Solomon code) along a data path. An error in any storage element that carries the structure can be detected at an appropriate check point where the structure is either discarded according to strict rules or reconstructed from the redundant data. Either way, the outage transient is limited to the transit time of this particular structure and as such is clearly of "low severity." It is worth noting that some packets may be considered highly critical and their loss may lead to larger outage. These statistical occurrence rates must be added to the "high-severity" category.

It is important to note that virtually every mitigation involves a HW and a SW component. A dropped packet must be accounted for. While an ECC detection/correction block in an ASIC can operate at data path speeds, there still remains the outstanding problem of having the software distinguish an upset from a persistent hardware bit fault. This requires an interrupt, counter and timer in SW. In addition, if the data is in a static table, SW must rewrite the correct value back so that the error will not appear persistent if the table is accessed frequently. A different interrupt will be required if a double bit error is detected, since this cannot be corrected by SECDED ECC. All of this functionality must be testable in the laboratory, so an error injection capability is required with supporting SW for a controlled test procedure. A fault in one of these SW handlers can make a single bit memory upset appear as a "high-severity" error if it causes the SW to crash. Although a system may have only a handful of discrete SRAM components, it is not uncommon for ASICs to have many hundred embedded SRAMs. This represents a significant amount of SW and HW behavior to create and verify. As a result, we rely heavily on internal design automation tools for automatic generation of ECC encoding and checking logic, error injection functionality, design verification test benches, documentation, and machine descriptions that generate driver code and diagnostic tests.

With all of the single bit memory upset accounted for, the remaining 2.5% of the total (about 14,000 FIT) presents a significant challenge. It is constituted mainly of single bit upsets in flip-flops (97%) and multi-bit upsets in embedded SRAM (3%). Predicted error rates provided by silicon manufacturers and in the literature have indicated that multicell memory error rates are becoming significant [7–9].

Adjacent SRAM cell upsets as high as 20% of the single bit FIT rate have been measured. All discrete SRAM components employ column multiplexing to ensure that logical bits of a word are not physically adjacent, so a multicell upset translates to multiple single bit errors that will be corrected by ECC. This same multiplexing technique is the best defense for embedded memories as well. However, limitations in the ASIC memory compilers and the unique requirements of an embedded application can pose limits on the physical distances of bits in a word. As a result, the only significant multi-bit SRAM error contribution to the model is from embedded SRAMs with limited or no column multiplexing.

At the present time, embedded SRAM multicell FIT rate estimation is very coarse due to the limited experimental and simulated data characterizing multicell error probabilities versus distance and its dependence on the particular topology of the memory cell and the fabrication process. Although the total multi-bit soft error rate is significant relative to the target "high-severity" specification limit, it is made up of hundreds of small contributions from the many embedded SRAMs. Our approach is to maximize the column multiplex ratio to obtain the maximum possible logical bit separation and then take advantage of the double error detection capability of the SECDED ECC function to trigger a reset as a common generic response. We support this with localized reset islands, in which a subset of hardware can be isolated and reset under software control without impacting the active state of the surrounding circuitry. Focusing our design effort into a generic, fast, localized reset for each chip avoids the complexity explosion of dealing with the many unique memory instances. The target for the fast reset is less than 10 ms (to achieve the "low-severity" outage class), although this cannot be verified until the entire software system is in place. Hence, it is conservatively allocated to the "medium-severity" class during the design phase.

10.3.2 Flip-Flops

Flip-flop soft error rates are receiving considerable attention in the industry as they have been steadily increasing with each step in ASIC technology evolution. Flip-Flop upsets result in unpredictable system behaviors and are likely responsible for the observations often described as single event functional interrupts (SEFI). At approximately 14,000 FIT for the line card, they represent a significant challenge to eliminate since they are to a first approximation contributed by millions of unstructured unique storage elements. Simply providing detection of an error is a huge challenge. A first pass analysis can identify some regular structures such as state machines and software configuration registers that can have parity detection added. Another pass can identify state storage that is included in other forms of detection such as packet check sequences (additional bits that accompany packets for error recovery) and parity protected buses. Maximizing the use of this type of circuitry is part of our design guidelines. These techniques can make approximately 1–5% of the flip-flop soft errors detectable. The generic local reset response (as used to

respond to multi-bit memory errors) is used to contain the outage to the low or medium classification bin.

Further reduction in flip-flop error rates requires more invasive strategies. Adding redundant logic such as TMR (triple modular redundancy) can triple the gate count required to implement a function. The remaining mitigation strategy is to substitute flop-flops with reduced soft error rates such as the DICE latch [10] into the design. Unfortunately, DICE latches are significantly larger than normal flip-flops and may also bring timing performance and power dissipation penalties. However, the total circuit impact is much less than the full triplication of voting circuits.

DICE latch soft error rates are difficult to predict today, as accurate tools for carrying out the 3D charge collection analysis on complex structures are not mature, and measured data are very rare. Other novel flip-flop structures have been proposed [11] that make use of scan test circuitry residing in some flip-flop cell designs. In mission mode, these extra latch elements can be configured into shadow storage to provide the reference state to detect and, with some additional circuitry, correct an upset. Since these approaches would require significant changes in ASIC test and clocking methodology, they are not yet available to us in a standard design flow. It is hoped that research in this area will yield more resilient and low impact methodologies that can be deployed.

10.3.2.1 Flip-Flop Error Masking

Another important area of research has been the analysis in which upsets predicted from an elemental circuit model actually result in an impact on system behavior [12, 13]. It turns out that there are many reasons why a flip-flop state upset may not affect system operation. These are summarized as follows:

Temporal masking: If an upset occurs in the slave stage of a flip-flop while it is transparent, then the value is immediately corrected by the master. Also, if an error occurs late in the clock cycle, then the incorrect value may not have time to propagate through the downstream logic and be captured by the next stage flip-flops.

Logical masking: If the state of a flip-flop is changed, the downstream combinatorial logic which is dependent on other state variables may prevent the incorrect value from propagating. For example, a multiplexer selects only one of its input branches, so an error on one of the other inputs would not propagate.

Functional masking: In a complex integrated circuit, even if an upset in one flip-flop is captured by downstream flip-flops, this may not result in observable change on the outputs. Even if the outputs do change, the system response may be benign. For example, a soft error in the random-number generator in a RED (Random Early Discard) circuit would be considered benign since it would result only in a different random sequence. The possibility that all the features within a chip are enabled is rare, so soft errors that occur in a disabled unit are functionally masked.

Predicting system reliability and making decisions regarding mitigation techniques require these masking factors to be accurately quantified. For general purpose ASICs, large parts of the design come from synthesized RTL (register transfer

level) code specific to the application, thus it is not generally possible to obtain the masking factors by extrapolation from existing circuits. Furthermore, masking factors can vary significantly depending on the type of circuit and its mode of operation. Fault injection simulation is a practical means for determining the flip-flop SER masking factors for an ASIC. Since digital simulation is the workhorse tool for the functional verification of ASICs, it is often possible, with a small incremental investment, to overlay fault injection simulations on top of the existing verification environment.

For a large digital ASIC, with simulation-based techniques, it is not possible to perform exhaustive fault injection – neither in time (every clock cycle) nor in space (every flip-flop). Statistically valid results can be obtained through sub-sampling in both dimensions (time and space). With care, these statistical masking results can be used in de-rating raw flip-flop FIT rates and predicting system reliability. Furthermore, statistical masking data can be helpful in optimizing mitigation strategies.

10.3.2.2 Error Classification

In order to discuss the effect of flip-flop upsets, it is helpful to further de-compose the possible effects of a soft error event in terms of the observed effect and the associated outage.

It is important to note that in many of the cases where the effect of an SEU is benign (rows 3–5 in Table 10.3), the sequence of vectors at the output pins of the device may be very different due to the effect of the SEU (e.g., an interrupt may be triggered, results may be delayed or re-ordered). Any approach that attempts to determine masking factors through a purely structural or static analysis of the circuit is unable to identify these opportunities for masking because it lacks application knowledge.

10.3.2.3 Device Model, Fault Model, and Stimulus

Fortunately, the logical and functional masking effects of SEUs can accurately be determined by simulating RTL level models. The gate-level mapping of any combinatorial cloud must be equivalent to the RTL description, thus the effect of an upset on any of its inputs will be the same. Of course, temporal masking effects and SET (single event transient) masking effects can only be determined accurately with detailed gate-level models. Generally, RTL simulations run an order of magnitude faster than gate-level simulations and require significantly less workstation memory, thus making it possible to perform a larger number of runs.

During logic synthesis, flip-flops are inferred from the RTL code. In order to perform fault injection simulations on an RTL model, the actual flip-flops in the design must be identified in the RTL code. One way of doing this is to extract the list of flip-flops from a matching gate-level netlist and then map them back to

Table 10.3 Classification of the effects of SEUs in ASICs

	Masking effect	Outage	Description
1	Logically masked	None	The value in the upset flip-flop does not propagate to any other flip-flops in the design. The state of the inputs to downstream combinatorial logic causes the upset value to be masked. For example, the upset flip-flop could be feeding an unselected branch of a mux
2	Functionally masked	None	The value in the upset flip-flop does propagate to some number of downstream flip-flops in the design; however, the sequence at the primary outputs was identical to that which it would have been had no upset occurred. Certain internal state points may be temporarily or permanently modified, but they do not affect any of the outputs for the given application (during the period of time under consideration)
3	Corrected	None	The upset was detected and corrected by mitigation circuitry such as ECC. There may be a log of the event and software intervention may be required to remove the corrupted data. The exact output sequence at the output pins may be different due to the error logging and in some cases temporary reduced performance; however, in the context of the application, the result is equivalent
4	System benign	Low	The upset caused the circuit to temporarily operate outside of its specification; however, the effect of this deviation is acceptable in the context of the system (at the low frequency of soft error events). For example, in a video application a single pixel in a video stream may be corrupted, in a routing application, the quality of service (QoS) guarantees are temporarily violated or in a micro-processor, a speculative branch may be mis-predicted causing a minor performance degradation
5	Detected – low outage	Low	The upset was detected and caused a minor outage (e.g., single packet drop in a router). This might arise when non-critical data are protected by parity. This type of event would generally be logged or counted and recovery might require software intervention
6	Detected – medium outage	Medium	The upset was detected and the recovery requires a medium-severity outage (e.g., the reset of a part of or a whole chip). This might arise when critical data are protected with parity or a hardware consistency check fails but the effect is well contained
7	Detected – high outage	High	The upset was detected and the recovery requires a high-severity outage (e.g., the reset of an entire card or major unit). This might arise when critical data are protected with parity or a hardware consistency check fails and multiple components are affected
8	Undetected fault	Unrecoverable	The effect of the upset was not detected by the circuit and resulted in a failure (data corruption, stuck network traffic). In a router, this might be the case of a device causing all incoming traffic to be silently dropped (“black hole”)

the RTL model. Obtaining a complete correspondence between the flip-flops in a final gate netlist and the RTL model can be difficult because sequential logic may be added as part of the back-end processing (Design for Test), flip-flops may be hidden in hard macros, and synthesis tools may prune flip-flops through logic optimization. So long as the purpose of the simulations is to draw statistical conclusions about the masking factors, if the number of unmapped flip-flops remains a small fraction of the total, this is still possible.

When considering the effect of SEUs on flip-flops, the basic fault model is that the value in a single flip-flop is inverted (e.g., changed from 1 to 0, or vice versa) at a point in time when the device is operating in its typical mode. In an ASIC, the physical location of flip-flops is determined by a placement tool and it is possible for multiple flip-flops to be adjacent such that a single event would upset multiple bits. With placement information, it is possible to refine the fault model to consider the effect of an event on adjacent flip-flops in close proximity; however, this is beyond the scope of what is considered in this section.

Care is required when selecting stimulus for fault injection simulations. To obtain results that correspond to what will be observed when a large population of systems is deployed in the field, the stimulus must be representative of "typical" applications. Often, the stimulus used for functional validation is crafted to exercise stressful logic states and may not be representative of typical system usage. A safe lower bound on the masking factors can be obtained by selecting a stimulus that represents a maximum level of activity. When compute capacity permits, a suite of simulations can be run to determine the variance of the masking factor due to application and load.

10.3.2.4 Results Checking

There are two broad approaches to determining the effect of an injected SEU in simulation. One of these is to compare the output pattern to a reference copy of the design where no upset occurs as shown in Fig. 10.4. Although this approach is easy to implement and can be generalized, it is not possible to differentiate between cases where the output sequence was different but benign in the system context (rows 3–5) in Table 10.3. One can only establish that the upset produced an observable difference, but not whether the different behavior was acceptable.

The alternative approach relies on a self-checking functional simulation environment to detect and report errors as shown in Fig. 10.5. Generally, the checking in a complex ASIC simulation environment is done using a combination of techniques (Table 10.4).

The advantage of using a functional simulation environment is that the correctness of the device is verified at a higher level of abstraction. This means that at the vector level on the outputs, the sequence may be significantly different as the result of the SEU, but the behavioral model has the knowledge to determine whether the modified sequence was acceptable.

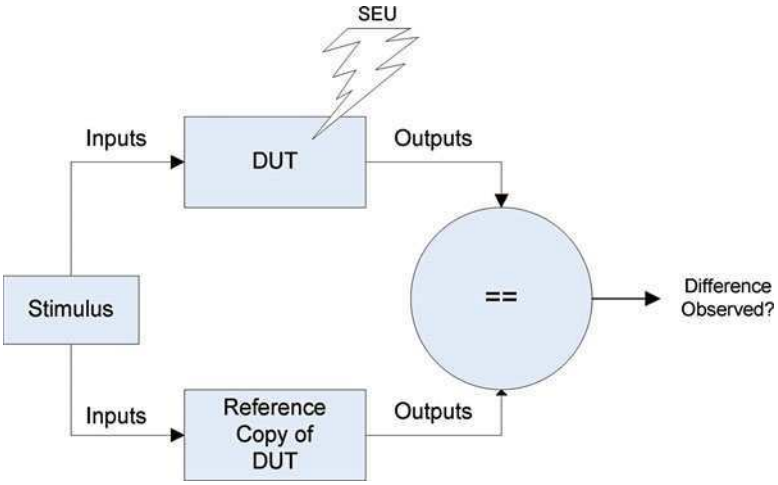


Fig. 10.4 Fault injection with a reference copy of design

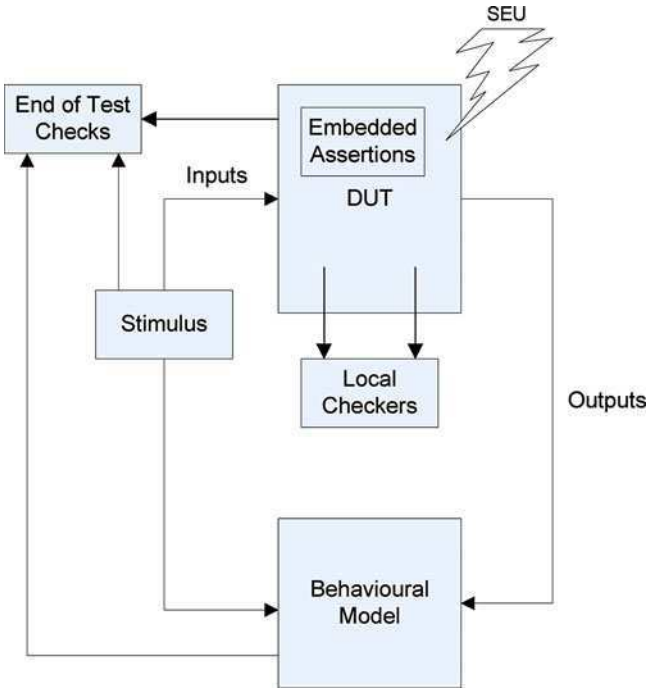


Fig. 10.5 Fault injection with a functional simulation environment

Table 10.4 Typical components of an ASIC functional simulation environment

Component	Description
Behavioral model	A model containing a scoreboard that validates the correctness of the outputs at a higher level of abstraction (e.g., transactions and packets). Usually, this type of model treats the device under test (DUT) as a black box
Local checkers	Checkers that monitor signals internal to the DUT and ensures that the behavior of a small portion of the circuit is correct
Embedded assertions	These assertions generally specify low-level rules that must hold true inside the design (e.g., if there is a request, it must be acknowledged within N clocks)
End of test checks	Certain aspects of design behavior can best be checked at the end of a simulation (e.g., performance and checking for memory leaks)

When performing fault injection simulations, the thoroughness of checking in the simulation environment can influence the measured functional masking. Care must be taken to ensure that the simulation checking is sufficiently strict to detect any deviation from normal behavior that would be detected in the real system. The implicit assumption when performing simulation-based masking analysis is that simulations which "pass" represent logical or functional masking and that they do not represent undetected faults.

If the simulation environment contains local checkers and embedded assertions, care must be taken not to equate all failures flagged by these checkers as observable SEUs. Failures must be analyzed in order to classify the actual outcome in the system. In most cases, if the only result of a fault injection is an error from a local checker, the fault is likely functionally masked downstream in the circuit (row 2 in Table 10.3).

10.3.2.5 ASIC Case Study

In this section, we show some results obtained from performing fault injection simulations on a large digital ASIC (1.2 M flip-flops, 25 Mbits of embedded memory) in a router application. Almost all the embedded memory on this device have SECDED ECC protection so that memory SEUs produce no outage. Certain large embedded memories representing approximately 75% of the memory bits can gracefully handle MBEs. A small set of memories are parity protected; however, the handling of parity errors is graceful. The MBE failure rate for the remaining memories is very small, thus flip-flop soft errors represent the primary failure mechanism. The raw flip-flop failure rate is relatively large and thus accurate determination of masking factors is critical to meeting system reliability targets.

Some basic techniques for flip-flop SER mitigation are used:

- Static configuration information is parity protected.
- The encoding of state-machines allows SEU detection.
- The payloads are protected by a CRC code as they transit through the device.

The benefit of these design techniques can only be quantified with fault injection simulations. The simulations were performed using the full-chip simulation environment, which consisted of behavioral models that check the contents of the data and control interfaces. Extensive end-of-test checks provide good detectability of latent errors (e.g., memory leaks and incorrect accounting).

Thousands of simulations were performed. During each simulation run, a flip-flop was selected randomly (spatial sub-sampling) and the time of the injection was selected randomly during the window of time when the device was fully active (temporal sub-sampling). If the simulation ran to completion without any difference being observed by the simulation environment, then the SEU was deemed to be either logically or functionally masked. The masking factor converged quickly and the result obtained after hundreds of simulations was the same as that obtained after thousands of simulations: *only one out of seven fault injections resulted in difference detected by the simulation environment*.

The set of runs where a difference was detected by the simulation environment was studied in further detail and the results were classified by the categories shown in Table 10.3. These results are shown in the following pie chart. The process of classifying failing runs was manual and thus not every failing run could be analyzed. When performing system level analysis, in order to be safe, all simulations which were not successfully classified were assumed to produce a high-severity outage.

It is interesting to note that of the runs where the simulation environment observed a difference in behavior, nearly half of these (48% from the *System Benign* and *Detected – Low Outage* categories) result in either no outage or a low-severity outage. Using static circuit analysis techniques, it would not be possible to identify those outcomes where the output sequence is significantly different after the upset.

The fault injection simulations provide two key pieces of data. First, an overall SEU de-rating factor showing that, in this particular case, six out of seven flip-flop SEUs produce no measurable change to the device behavior when it is operating in mission mode. Second, of the one in seven of the SEU upsets that produces a perturbation in the behavior, half of these are detected by an existing mechanism and results in a low-severity outage.

10.3.2.6 Temporal Masking Analysis

In addition, we have carried out a slack time analysis on flip-flop fan in. The methodology used to perform this analysis was based on analyzing the gate-netlist to determine the connectivity and standard delay format (SDF) files to extract the timing. The ratio of the maximum slack path to the clock period for each flip-flop fan in tree is then summed across the chip. The results indicated that approximately 10% of SEUs fail to meet the next stage setup window for the best case timing corner. At the worst case timing corner, our analysis showed that about 20% of the upsets are temporally masked. An upset is only considered to be temporally masked if it does not propagate to *any* of the downstream flops, thus the effect is limited by

the timing on the fastest path. The reality is more complex, because an upset may actually propagate to some downstream flops and be temporally masked to others; however, this quickly becomes difficult to analyze. Thus, we chose a conservative definition of temporal masking using best case timing on the shortest paths for a masking factor of 0.9.

We recognize that these masking factors cannot be generalized to all our ASICs and so we are continuing to refine our analysis of functional and temporal masking with additional chips representing different design applications. Ultimately, we expect this type of analysis to become part of our regular design flow. Applying these masking factors to the remaining flip-flops provided sufficient confidence that we would meet our soft error target without further design changes.

10.3.3 Model Results

The model results are summarized in Table 10.5 with temporal masking of 0.9 and functional masking of 0.16 applied. The rate predicted from the "Unrecoverable" class is slightly above specification but does not warrant any design intervention based on the results from fault injection simulation, which show that approximately half of the flip-flop SEUs that are visible actually result in a low-severity outage. As

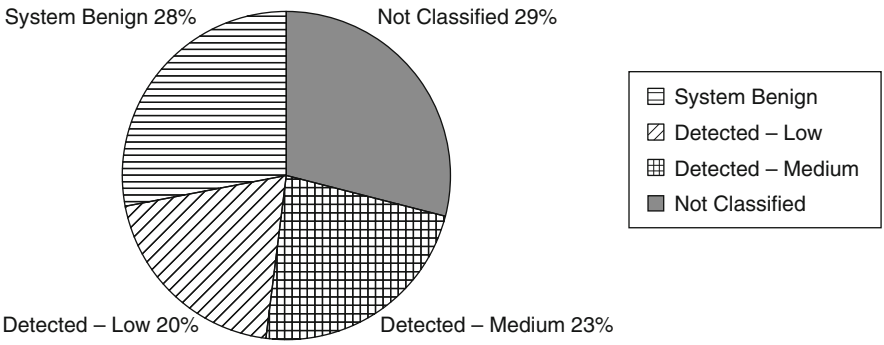


Fig. 10.6 Classification of failure types resulting from fault injection

Table 10.5 Predicted soft error rates relative to the target specification

Class	Maximum recovery time (s)	Maximum outage rate (FIT)	Predicted outage rate (FIT)
No outage	0	Unlimited	533,000
Low	0.01	30,000	1,480
Medium	3	10,000	4,770
High	180	1,000	820
Unrecoverable	1,000	1,000	1,120

mentioned earlier, the second order effects that dominate the FIT rate calculation in the "high" and "unrecoverable" category have very little hard experimental data to establish confidence intervals. Obtaining greater accuracy in flip-flop and multi-bit memory soft error rates from ASIC vendors is highly important.

10.4 Verifying Soft Error Performance

The combination of modeling uncertainties and the considerable hardware and software complexity that make up the prediction of the soft error performance of a system mandate some kind of verification strategy to ensure that a new product is not deployed with a significant vulnerability. The nature of soft error behavior is such that if a problem exists, you may not discover it until a significant number of systems have been shipped and in service. To avoid this, we employ accelerated testing using a neutron beam facility. The goals of this test are as follows:

- Identify bugs in the detection and response mechanisms;
- Identify rogue components that do not meet their specified FIT rates or display latch-up behavior, which would eliminate them as eligible components;
- Verify that the system level soft error performance specification has been met – particularly for the "high-severity" class.

Neutron irradiation is a convenient method for performing this type of system analysis since neutrons are a dominant source of upset in nature at ground level, and it is possible to obtain industrial access to several facilities in the world that can generate a large area beam of fairly uniform flux density. Since neutrons have very high penetration, a beam that is wide enough to expose the entire active portion of the system can have all of its hardware bombarded simultaneously. By placing supporting test equipment outside the test area, we can effectively measure the behavior of a router passing traffic in its native operating state. We can then adjust the beam flux in order to generate an accelerated test of the order of 10^6 (~1 year per minute of beam time) relative to an assumed neutron flux rate of 14 neutrons/cm^2/h at sea level. This is sufficient to measure the relatively small failure rates targeted for "high-severity" errors in a reasonable time interval while keeping the more frequent "low-severity" rate below one event per second, so that we do not overload the control path with interrupts. For our measurements, we have used both the quasi-monoenergetic neutron beam facilities at the Svedberg Research Lab at the University of Uppsala, Sweden, and the broad spectrum facility at TRIUMF at the University of British Columbia, Canada.

We prepared two sets of software loads. The first is a diagnostic test that simply runs continuous patterns on all memories on a card and detects, counts, and corrects all errors. Statistics gathered from this test verify that the anticipated component error rates are as expected and helps isolate any rogue components not performing as expected. The second SW load is the mission mode operating system of the router. Traffic is run and features are turned on that attempt to load and stress as

much of the hardware as possible in a realistic application scenario. The beam is turned on and the system is monitored for SW crashes and any other system error messages. Log files are gathered for diagnosis and isolation of SW problems that need to be fixed.

Since there are many thousands of recovery routines required to respond to the possible soft error interrupts generated by the hardware, and some are quite difficult to test in a normal test laboratory environment even with the error injection functions built into the ASICs, it is not uncommon for SW bugs to hide in these areas of the code. Many may be associated with very rare error syndromes. With this methodology, we are able to detect the most important bugs and continuously improve the performance of the system in subsequent runs.

Figure 10.7 shows the results from a particular run for a group of CISCO 12000 SIP 600/601 series line cards measured at the TRIUMF test facility. The chart shows the results for the same hardware running the two different operating systems, IOS and IOS-XR, which are available on the product. Error bars are used to illustrate the 90% certainty range in the FIT rate measurements based on chi square statistics. When fewer failures are seen over a given exposure time, the uncertainty in the measurements rise.

The difference in the two measurements in Fig. 10.4 illustrates the role that the SW plays. Both software loads were configured to perform the same network tests and both were clearly demonstrating very high reliability (less than 1,000 FIT), at the time these measurements were performed. However, the IOS-XR software load was a new product on the SIP600/601 and this was its first exposure to neutron beam testing, whereas the more mature IOS load was in its third test campaign. Failures found in the tests are used to drive software improvements that continually

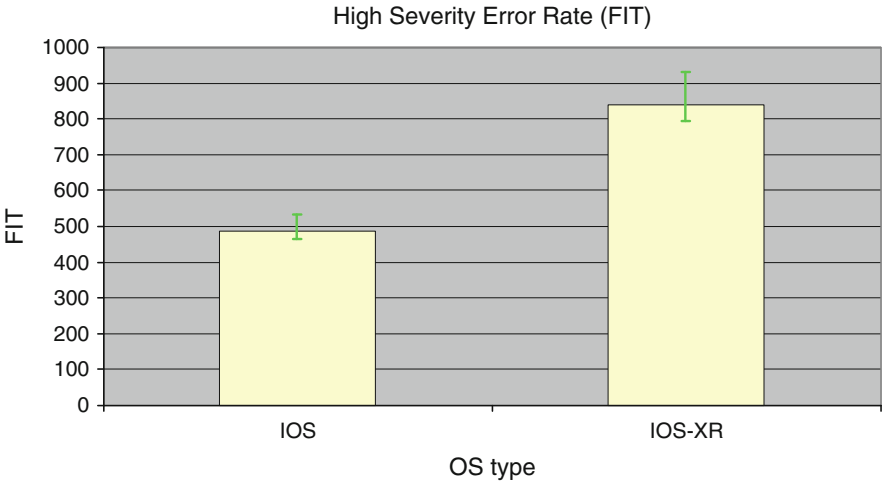


Fig. 10.7 Measured Cisco 12000 SIP 600/601 accelerated soft error rates. Soft error rates are normalized to New York City sea level reference for the "high-severity" category running the IOS and IOS-XR operating systems. *Error bars* show 90 confidence levels

lower the high-severity outage frequencies with successive test campaigns. Since the hardware is identical, we can expect both operating systems to achieve ultimately the same performance as the IOS-XR software catches up in its maturity.

It is important to note that these measurements are only for the neutron component of the full system soft error rate. Since this product is constructed largely of silicon from the 130-nm generation technology node, we expect an additional 30% contribution from alpha-induced errors based on vendor supplied component data. If we assume that the errors found in neutron testing are statistically representative of the errors that would be induced by alpha particles, then we would need to multiply the above results by 1.3 to obtain a total high-severity error rate in the field. Therefore, this line card is achieving the target specification running IOS and is slightly above specification running IOS-XR. We expect the IOS-XR rate to drop in subsequent measurements as we fix the problem sources that were observed in this run.

10.5 Conclusions

We have presented a methodology for relating single event upset rates to high availability requirements for a complex system of ASICs and software. By considering component soft error FIT rates and resulting outage times in the system together, we are able to construct a unified specification that can be used to drive design decisions. This provides clarity in responding to issues such as where and in what quantity to deploy resilient flip-flops, or whether a targeted fast recovery mechanism or a slower generic reset is sufficient to respond to a particular error when it is detected. This supports both ASIC and software design during the product development stage.

For the case of an Internet core router, we have examined the end-user metrics of network availability, quality of service, and the service lifetime of a replaceable component such as a line card to develop the specification. We identified important breakpoints in outage time intervals that are relevant to the end-user experience that enable the error rate contributions from different hardware components to be binned according to the error recovery mechanism that they trigger. We found that in each of the bins, a different aspect of the end-user requirement applies, with the most demanding requirement being for outages greater than 3 s.

To meet this requirement in a next-generation line card, the analysis required a detailed model including what were previously considered second-order effects (multi-bit memory errors and flip-flop upsets). Further, a deeper analysis of temporal, logical, and functional masking of flip-flop upsets was required to determine if millions of flip-flops would have to be replaced by larger and more power hungry DICE latch cells. The result was that the specification could be met without these substitutions for this level of integration at 65 nm. However, it is clear that we will need much better tools to support this analysis in future product developments.

Finally, we presented a measurement technique using large area neutron beam radiation to provide an accelerated test of the hardware and software together executing their mission mode application. Using this method, we can directly measure the performance against the key metrics in the specification as well as identify and fix problems long before they are seen in Internet service provider networks. With this technique, we are able to demonstrate that systems that were previously designed to this requirement have indeed achieved their intended performance.

Acknowledgments The authors would like to thank D. Mah, W. Obeidi, M. Pugi, M. Kucinska, A. Agrawal, M. Ruthruff, P. Narayan, J. Bolduc, N. Taranath, and P. Therrien of Cisco Systems for their contributions to the neutron beam testing effort. We would also like to thank M. Olmos and D. Gauthier at iROC Inc, as well as E. Blackmore at TRIUMF and A. Prokofiev at TSL, for their critical support at the neutron test facilities. Finally, we would like to thank David Jones at Xtreme EDA for his contributions to the error masking investigations. The TRIUMF facility is supported by funding from the National Research Council of Canada.

References

1. D. Miras, "A survey of network QOS needs of advanced internet applications," 2002, Internet2 – QoS working group, pp. 30–31, http://qos.internet2.edu/wg/apps/fellowship/Docs/Internet2AppsQoSNeeds.html.
2. "Transmission systems and media, digital systems and networks," ITU-T Recommendation G.114, pp. 2–3, approved May 2003.
3. D. Katz, D. Ward, "Bidirectional Forwarding Detection," IETF Network Working Group, Internet-Draft, http://tools.ietf.org/html/draft-ietf-bfd-base-00.
4. V. Paxson , M. Allman, "Computing TCP's Retransmission Timer," IETF Network Working Group, RFC 2988, p. 1, http://www.ietf.org/rfc/rfc2988.txt.
5. "Network performance objectives for IP-based services," ITU-T Recommendation Y.1541, approved Feb. 22, 2006.
6. C.L. Chen, M.Y. Hsiao, "Error-correcting codes for semiconductor memory applications: a state-of-the-art review," IBM Journal of Research and Development, vol. 28, pp. 124–134, Mar. 1984.
7. E. Ibe et al, "Spreading diversity in multi-cell neutron-induced upsets with device scaling," Custom Integrated Circuits Conference, San Jose CA, pp. 437–444, Sept. 2006.
8. N. Seifert et al, "Radiation-induced soft error rates of advanced CMOS bulk devices," Int. Reliability Physics Symposium, San Jose, CA, pp. 217–224, 2006.
9. Y. Tosaka et al, "Comprehensive study of soft errors in advanced CMOS circuits with 90/130 nm technology," Int. Electron Devices Meeting, San Francisco, CA, pp. 38.3.1–38.3.4, Dec. 2004.
10. T. Calin, M. Nicoaidis, R. Velazco, "Upset hardened memory design for submicron CMOS technology," IEEE Transactions on Nuclear Science, vol. 43 no. 6, pp. 2874–2878, Dec. 1996.
11. S. Mitra, N. Seifert, M. Zhang, K. Kim, "Robust system design with built-in soft-error resilience," IEEE Computer, pp. 43–52, Feb. 2005.
12. H.T. Nguyen, Y. Yagil, N. Siefert, "Chip-level soft error estimation method," IEEE Transactions on Device and Materials Reliability, vol. 5, no. 3, pp. 365–381, Sept. 2005.
13. N. Wang, J. Quek, T. Rafacz, S. Patel, "Characterizing the effects of transient faults on a high-performance processor pipeline," Proc. of the Intl. Conf. on Dependable Systems and Networks, pp. 61–70, 2004.

Index

A
ABFT. *See* Algorithm-based fault tolerance
Accelerated tests, 168, 192
Acceptance test. *See* Decider
Active-stream/redundant-stream simultaneous multithreading, 262
Algorithm-based fault tolerance, 275, 278
Alpha activity, 56
Alpha particles, 2, 3, 22, 29
Alpha spectra, 58, 59
Alternate. *See* Variant
Altitude, 63, 64, 72
Ambipolar diffusion, 83
Arithmetic codes, 234
ASIC, 86, 287
Assertions, 264
Atmospheric neutrons, 77, 80
Atmospheric spectrum, 98, 100
Autonomous emulation, 156, 160, 163

B
Babbage, C., 273
Barometric pressure, 63, 64
Basic block, 281
Berger code, 234
Best effort recovery, 264
BFD. *See* Bidirectional forwarding distribution
Bidirectional forwarding distribution, 291
Bit-flip fault model, 145, 153
Boro-phospho-silicate glass, 6, 69
Bragg peak, 62

C
Charge collection, 39, 83
Charge zone, 83
Checkpoint(ing), 262, 276
Checkpointing overhead, 277
Checksum matrix, 278
Chisquare distribution, 74
Collapsed transient currents analysis, 92
Combinational logic, 20
Commercial-off-the-shelf components, 254
Comparison vector, 274
Computation duplication, 255
Consensus, 273
Context switch, 262
Control flow check, 266
Control flow graph, 265
Correction, 281
Cosmic radiation, 2, 5, 6
Coulomb scattering, 59
Critical charge, 3, 39, 45, 57, 86
Cross section, 65–67, 71, 170, 173
Current injection simulation, 92
Current pulse, 83, 84, 90
Current shapes, 92
Cyclic redundancy check codes, 218, 219
Cyclotron, 167, 170, 182

D
DASIE. *See* Detailed analysis of secondary ion effect
Data diversity, 262
Decider, 273–275
Decision point, 273
Decoding operation, 282
Decontamination, 214
Defects per million, 288
Delay buffer, 262
Dependability evaluation, 142, 143
Design diversity, 273
Detailed analysis of secondary ion effect (DASIE), 81, 84, 85, 91, 93
DICE, 229
Differential flux, 57, 63, 64, 67
Dffusion-collection model, 78, 101

Digital SET, 43
Diversity factor, 262
Double sampling, 238, 239, 246
Double sampling time borrowing, 246
DPM. *See* Defects per million
DRAM. *See* Dynamic random access memory
DSTB. *See* Double sampling time borrowing
Duplication and checksum, 280
Duplication and Hamming code, 281
Dynamic cross-section, 180, 192
Dynamic random access memory (DRAM), 3, 14
Dynamic SEE analysis, 127

E
Early fault classification, 159
ECAM. *See* Error content addressable memory
ECC. *See* Error correcting code
EDR. *See* Electrical derating
Elastic interaction, 31
Electrical derating, 117, 118
Emulation-based fault injection, 148, 151, 152
Enterprise, 9
Equivalence class, 92
Error content addressable memory, 163
Error correcting code, 206, 288
Error-correction coding, 14, 15
Error detection by diverse data and duplicated instructions, 263
Error-detection by duplicated instructions, 257

F
Failure (fault classification), 153
Failure rate, 100
Failures in time (FIT), x, 2, 58, 68, 72–74, 288
Fault, 277
Fault classification, 153
Fault dictionary, 154
Fault injection, 92, 93, 168, 181, 192, 193, 199, 305
Fault list, 153
Fault secure, 233, 234
Fault-tolerant processor-attached processor architecture, 274
Faulty circuit, 150
FDR. *See* Functional derating
Fibonacci, 263
FIT. *See* Failures in time
Flip flops, 16, 287, 299
FPGA, 180, 188, 200
Functional derating, 118, 127
Functional failure, 112
Functional masking, 300

G
Geomagnetic rigidity, 64
Golden circuit, 150
Golden part, 57

H
Hamming code, 206, 209
Hardened storage cells, 227
Hardware fault injection, 141
Hera, 8
High energy neutron spectra, 64
H matrix, 208, 209
Hsiao code, 209, 210

I
Insert sort, 263
Instruction-level computation duplication, 256
Instruction re-execution, 258
Instrument, 148, 149, 159, 160
Interaction probability, 30
Internet core router, 310

J
JEDEC. *See* Joint Electron Device Engineering Council
Joint Electron Device Engineering Council (JEDEC), 55, 56, 75

L
Laser, 167
Latches, 16
Latch-up, 49
Latent (fault classfication), 154
Layout, 86, 87, 91
LDR. *See* Logic derating
LET. *See* Linear energy transfer
Linear accelerator, 167, 170
Linear cyclic code, 221
Linear energy transfer (LET), 35, 59, 171
Logical fault injection, 146
Logical masking, 300
Logic derating, 118, 125, 132
Lost of sequence (fault classification), 154
Lzw compression, 263

M
Mask flip-flop, 149
MBE. *See* Multiple bit errors
MBU. *See* Multiple bit upset
MCU. *See* Multiple cell upsets
Mean time between failures, 288
Modified Hamming code, 209
Monte Carlo, 77, 78, 80, 81, 86, 101

m-out-of-n code, 234
MTBF. *See* Mean time between failures
Multiple bit errors, 290
Multiple bit upset, 57, 109
Multiple cell upsets, 13, 57, 84, 109
Multiple event transients, 77, 86

N
Netlist, 91, 94
Neutron beam testing, 309
Neutrons, 2
Neutron-silicon interactions, 81
Neutron spallation, 65
N-modular redundancy, 274
NMR. *See* N-modular redundancy
Noise margins, 254
Nonelastic interaction, 31
Nuclear database, 80–82, 86
Nuclear reaction, 77, 78, 80, 81, 86, 87, 90
N-version programming, 273
NVP. *See* N-version programming

O
Occurrence probability, 77, 90, 98

P
Parity code, 210, 233
Particle beams, 168, 181, 195, 197
Particle fluxes, 177, 189
Physical fault injection, 144
Poisson distribution, 73
Procedure-level computation duplication, 256
Procedure-level duplication, 258
Processor, 168, 169, 173, 177, 178, 181, 183, 186, 193
Program-level computation duplication, 256
Program-level duplication, 260
Pulse width, 106, 123
PW. *See* Pulse width

Q
QOS. *See* Quality of service
Quality of service, 289
Quantized delay model, 150
Quick sort, 263

R
Radiation environment, 29
Radiation ground testing, 168, 170, 172, 180, 185, 192, 196, 199
Radiation-hardened, 18
Radioactive impurities, 2, 4, 8, 22
Random fault injection, 287
RB. *See* Recovery blocks
Real-time SER testing, 6
Recoils, 33
Recovery blocks, 273
Recovery point, 275
Reed-Solomon codes, 219, 222, 223
Register transfer level, 300
Rollback recovery, 276
Roll-forward checkpoint, 276
Router line cards, 10
RTL. *See* Register transfer level

S
5 9's, 288, 293, 294
Satellite electronics, 3
SBE. *See* Single bit errors
SBU. *See* Single bit upset
Scaling trends, 10, 17, 19
SECDED. *See* Single error correction double error detection
Secondary ions, 80–83, 87, 88
SEE. *See* Single event effects
SEFI. *See* Single event functional interrupts
Selective procedure call duplication, 258
Self-checking design, 232
SER. *See* Soft error rate
SET. *See* Single event transients
SEU. *See* Single event upsets
Shallow trench insulators, 87
SIHFT. *See* Software-implemented hardware fault tolerance
Silent (fault classification), 153
Silicon-on-insulator, 12, 22
Simultaneous multithreading, 261
Single bit errors, 290
Single bit upset, 1, 2, 109
Single-error correcting codes, 210
Single error correction double error detection, 297
Single event effects (SEE), 1, 57, 103, 143, 167, 169, 170
  simulation, 128
Single event functional interrupts, 48, 111, 299
Single-event latch-up, 2, 21, 57, 169, 178, 185, 189, 290
Single event transients (SET), 2, 77, 105, 109, 123, 132, 142, 144, 150, 203, 301
  mitigation, 230
Single event upsets, 44, 77, 84, 108, 120, 133, 142, 144, 168, 169, 177, 203, 287, 310
Soft error, 111
Soft error masking, 287

Soft error rate, x, 2, 39, 56, 61, 65–68, 71, 74, 77, 112, 154
Soft error specification, 288
Software bug, 274
Software-implemented hardware fault tolerance, 254
Solar activity, 63, 64
Spatial redundancy, 262
SPCD. *See* Selective procedure call duplication
Spherical diffusion law, 88, 89
SPICE simulation, 77, 79, 90
Spread, 13
SRAM, 2, 11
Standby sparing, 275
State restoration, 158
Static cross-section, 180, 185, 197
Stop and retry recovery, 276
Superscalar microprocessors, 261
Syndrome, 206–209

T
TCP. *See* Transmission control protocol
TDR. *See* Temporal derating
Temporal derating, 118, 120, 132
Temporal masking, 300, 306
Test platform, 178, 180, 191, 194, 195, 199
Thermal neutrons, 6
Thermal neutron spectra, 70
Time quantum, 150
Time redundancy, 258, 260, 262
TMR. *See* Triple modular redundancy
Track, 37
Transient duration, 77
Transient pulses, 89
Transition detection time borrowing, 246
Transmission control protocol, 292
Triple modular redundancy, 274, 299

U
Unordered codes, 233

V
Variant, 260, 273
Version. *See* Variant
Very long instruction word processors, 258
Virtual duplex system, 260
VLIW. *See* Very long instruction word processors

W
Weibull function, 66